Développons en Java

Jean Michel DOUDOUX

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Développons en Java

Version 0.85 béta

du 22/11/2005

par Jean Michel DOUDOUX

Préambule

A propos de ce document

L'idée de départ de ce document était de prendre des notes relatives à mes premiers essais en Java. Ces notes ont tellement grossies que j'ai décidé de les formaliser un peu plus et de les diffuser sur internet d'abord sous la forme d'articles puis rassemblées pour former le présent didacticiel.

Celui−ci est composé de six grandes parties :

1. les bases du langage java
2. le développement des interfaces graphiques
3. les API avancées
4. le développement d'applications d'entreprises
5. les outils de développement
6. le développement d'applications mobiles

Chacune de ces parties est composée de plusieurs chapitres dont voici la liste complète :

- Préambule
- Presentation de Java
- Technique de base de la programmation Java
- La syntaxe et les éléments de bases de Java
- POO avec Java
- Les packages de base
- Les fonctions mathématiques
- La gestion des exceptions
- Le multitâche
- JDK 1.5 (nom de code Tiger)
- Le graphisme en java
- Les éléments d'interfaces graphiques de l'AWT
- La création d'interfaces graphiques avec AWT
- L'interception des actions de l'utilisateur
- Le développement d'interfaces graphiques avec SWING
- Le développement d'interfaces graphiques avec SWT
- JFace
- Les applets en java
- Les collections
- Les flux
- La sérialisation
- L'interaction avec le réseau
- L'accès aux bases de données : JDBC
- La gestion dynamique des objets et l'introspection
- L'appel de méthode distantes : RMI
- L'internationalisation
- Les composants java beans
- Logging
- La sécurité
- Java Web Start (JWS)
- JNI (Java Native Interface)
- JDO (Java Data Object)
- D'autres solutions de mapping objet−relationnel
- Java et XML
- Java 2 Entreprise Edition
- Les servlets
- Les JSP (Java Servers Pages)
- JSTL (Java server page Standard Tag Library)
- Les frameworks pour les applications web
- Java Server Faces

- JNDI (Java Naming and Directory Interface)
- JMS (Java Messaging Service)
- JavaMail
- Les EJB (Entreprise Java Bean)
- Les services web
- Les outils du J.D.K.
- Les outils libres et commerciaux
- JavaDoc
- Java et UML
- Des normes de développement
- Les motifs de conception (design patterns)
- Ant
- Maven
- Les frameworks de tests
- Des bibliothèques open source
- Des outils open source
- J2ME
- CLDC
- MIDP
- CDC
- Les profils du CDC
- Les autres technologies

Je souhaiterais le développer pour qu'il couvre un maximum de sujets autour du développement en Java. Ce souhait est ambitieux car l'API de Java est très riche et ne cesse de s'enrichir au fil des versions.

Dans chaque partie, les membres des classes décrites ne le sont que partiellement : pour une description complète de chaque classe, il faut consulter la documentation fournie par Sun au format HTML pour les API du JDK et la documentation fournie par les fournisseurs respectifs des autres API tiers.

Je suis ouvert à toutes réactions ou suggestions concernant ce document notamment le signalement des erreurs, les points à éclaircir, les sujets à ajouter, etc. ... N'hésitez pas à me contacter : jean−michel.doudoux@wanadoo.fr

Ce document est disponible aux formats HTML et PDF à l'adresse suivante : http://perso.wanadoo.fr/jm.doudoux/java/

Ce manuel est fourni en l'état, sans aucune garantie. L'auteur ne peut être tenu pour responsable des éventuels dommages causés par l'utilisation des informations fournies dans ce document.

La version pdf de ce document est réalisée grâce à l'outil HTMLDOC verison 1.8.23 de la société Easy Software Products. Cet excellent outil freeware peut être téléchargé à l'adresse : http://www.easysw.com

Remerciements

Je tiens à remercier les personnes qui m'ont apporté leur soutien au travers de courrier électronique de remerciements ou de félicitations.

Je tiens aussi particulièrement à exprimer ma gratitude aux personnes qui m'ont fait part de correctifs ou d'idées d'évolutions : ainsi pour leurs actions, je tiens particulièrement à remercier Vincent Brabant et Thierry Durand.

Notes de licence

La version la plus récente de cette licence est disponible à l'adresse : GNU Free Documentation Licence.

Marques déposées

Sun, Sun Microsystems, le logo Sun et Java sont des marques déposées de Sun Microsystems Inc.

Les autres marques et les noms de produits cités dans ce document sont la propriété de leur éditeur respectif.

Historique des versions

Version	Date	Evolutions
0.10	15/01/2001	brouillon : 1ere version diffusée sur le web.
0.20	11/03/2001	ajout des chapitres JSP et serialization, des informations sur le JDK et son installation, corrections diverses.
0.30	10/05/2001	ajout des chapitres flux, beans et outils du JDK, corrections diverses.
0.40	10/11/2001	réorganisation des chapitres et remise en page du document au format HTML (1 page par chapitre) pour faciliter la maintenance ajout des chapitres : collections, XML, JMS, début des chapitres Swing et EJB séparation du chapitre AWT en trois chapitres.
0.50	31/04/2002	séparation du document en trois parties ajout des chapitres : logging, JNDI, Java mail, services web, outils du JDK, outils lires et commerciaux, Java et UML, motifs de conception compléments ajoutés aux chapitres : JDBC, Javadoc, intéraction avec le réseau, Java et xml, bibliothèques de classes
0.60	23/12/2002	ajout des chapitres : JSTL, JDO, Ant, les frameworks ajout des sections : Java et MySQL, les classes internes, les expressions régulières, dom4j compléments ajoutés aux chapitres : JNDI, design patterns, J2EE, EJB
0.65	05/04/2003	ajout d'un index sous la forme d'un arbre hiérarchique affiché dans un frame de la version HTML ajout des sections : DOM, JAXB, bibliothèques de tags personnalisés, package .war compléments ajoutés aux chapitres : EJB, réseau, services web
0.70	05/07/2003	ajout de la partie sur le développement d'applications mobiles contenant les chapitres : J2ME, CLDC, MIDP, CDC, Personal Profile, les autres technologies ajout des chapitres : le multitache, les frameworks de tests, la sécurité, les frameworks pour les app web compléments ajoutés aux chapitres : JDBC, JSP, servlets, intéraction avec le réseau application d'une feuille de styles CSS pour la version HTML corrections et ajouts divers

0.75	21/03/2004	ajout des chapitres : le développement d'interfaces avec SWT, Java Web Start, JNI compléments ajoutés aux chapitres : GCJ, JDO, nombreuses corrections et ajouts divers notamment dans les premiers chapitres
0.80 0.80.1 0.80.2	29/06/2004 15/10/2004 02/11/2004	ajout des chapitres : le JDK 1.5, des bibliothèques open source, des outils open source, Maven et d'autres solutions de mapping objet−relationnel ajout des sections : Installation J2SE 1.4.2 sous windows, J2EE 1.4 SDK, J2ME WTK 2.1 compléments ajoutés aux chapitres : Ant, Jdbc, Swing, Java et UML, MIDP, J2ME, JSP, JDO nombreuses corrections et ajouts divers
0.85	22/11/2005	ajout du chapitre : Java Server Faces ajout des sections : java updates, le composant JTree nombreuses corrections et ajouts divers

Partie 1 : les bases du langage Java

Cette première partie est chargée de présenter les bases du langage java.

Elle comporte les chapitres suivants :

- Présentation : introduit le langage Java en présentant les différentes éditions et versions du JDK, les caractéristiques du langage et décrit l'installation du JDK
- Les techniques de base de programmation en Java : présente rapidement comment compiler et executer une application
- La syntaxe et les éléments de bases de Java : explore les éléments du langage d'un point de vue syntaxique
- La programmation orientée objet : explore comment Java utilise et permet d'utiliser la programmation orientée objet
- Les packages de bases : propose une présentation rapide des principales API fournies avec le JDK
- Les fonctions mathématiques : indique comment utiliser les fonctions mathématiques
- La gestion des exceptions : explore la faculté de Java pour traiter et gérer les anomalies qui surviennent lors de l'exécution du code
- Le multitâche : présente et met en oeuvre les mécanismes des threads qui permettent de répartir différents traitements d'un même programme en plusieurs unités distinctes pour permettre leur exécution "simultanée"
- JDK 1.5 (nom de code Tiger) : détaille les nouvelles fonctionnalités du langage de la version 1.5

1. Présentation

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C. Il existe 2 types de programmes en Java : les applets et les applications. Une application autonome (stand alone program) est une application qui s'exécute sous le contrôle direct du système d'exploitation. Une applet est une application qui est chargée par un navigateur et qui est exécutée sous le contrôle d'un plug in de ce dernier.

Ce chapitre contient plusieurs sections :

1.1. Les caractéristiques

Java possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès :

Java est interprété	le source est compilé en pseudo code ou byte code puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout). En effet, le byte code, s'il ne contient pas de code spécifique à une plate−forme particulière peut être exécuté et obtenir quasiment les même résultats sur toutes les machines disposant d'une JVM.
Java est indépendant de toute plate−forme	il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du byte code.
Java est orienté objet.	comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).
Java est simple	le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs (pour éviter les incidents en manipulant directement la mémoire), l'héritage multiple et la surcharge des opérateurs, ...
Java est fortement typé	toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.
Java assure la gestion de la mémoire	

	l'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.
Java est sûr	la sécurité fait partie intégrante du système d'exécution et du compilateur. Un programme Java planté ne menace pas le système d'exploitation. Il ne peut pas y avoir d'accès direct à la mémoire. L'accès au disque dur est réglementé dans une applet. Les applets fonctionnant sur le Web sous soumises aux restrictions suivantes dans la version 1.0 de Java : • aucun programme ne peut ouvrir, lire, écrire ou effacer un fichier sur le système de l'utilisateur • aucun programme ne peut lancer un autre programme sur le système de l'utilisateur • toute fenêtre créée par le programme est clairement identifiée comme étant une fenêtre Java, ce qui interdit par exemple la création d'une fausse fenêtre demandant un mot de passe • les programmes ne peuvent pas se connecter à d'autres sites Web que celui dont ils proviennent.
Java est économe	le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.
Java est multitâche	il permet l'utilisation de threads qui sont des unités d'exécution isolées. La JVM, elle même, utilise plusieurs threads.

Les programmes Java exécutés localement sont des applications, ceux qui tournent sur des pages Web sont des applets.

Les principales différences entre une applet et une application sont :

- les applets n'ont pas de méthode main() : la méthode main() est appelée par la machine virtuelle pour exécuter une application.
- les applets ne peuvent pas être testées avec l'interpréteur mais doivent être intégrées à une page HTML, elle même visualisée avec un navigateur disposant d'un plug in sachant gérer les applets Java, ou testées avec l'applet viewer.

1.2. Bref historique de Java

Les pincipaux événément de la vie de Java sont les suivants :

Année	Evénements
1995	mai : premier lancement commercial
1996	janvier : JDK 1.0
1996	septembre : lancement du JDC
1997	février : JDK 1.1
1998	décembre : lancement de J2SE et du JCP
1999	décembre : lancement J2EE
2000	mai : J2SE 1.3
2002	J2SE 1.4
2004	J2SE 5.0

1.3. Les différentes éditions et versions de Java

Sun fourni gratuitement un ensemble d'outils et d'API pour permettre le développement de programmes avec Java. Ce kit, nommé JDK, est librement téléchargeable sur le site web de Sun http://java.sun.com ou par FTP ftp://java.sun.com/pub/

Le JRE (Java Runtime Environment) contient uniquement l'environnement d'exécution de programmes Java. Le JDK contient lui même le JRE. Le JRE seul doit être installé sur les machines où des applications Java doivent être exécutées.

Depuis sa version 1.2, Java a été renommé Java 2. Les numéros de version 1.2 et 2 désignent donc la même version. Le JDK a été renommé J2SDK (Java 2 Software Development Kit) mais la dénomination JDK reste encore largement utilisée, à tel point que la dénomination JDK est reprise dans la version 5.0. Le JRE a été renommé J2RE (Java 2 Runtime Environment).

Trois éditions de Java existent :

- J2ME : Java 2 Micro Edition
- J2SE : Java 2 Standard Edition
- J2EE : Java 2 Entreprise Edition

Sun fourni le JDK, à partir de la version 1.2, sous les plate−formes Windows, Solaris et Linux.

La version 1.3 de Java est désignée sous le nom Java 2 version 1.3.

La version 1.5 de Java est désignée officiellement sous le nom J2SE version 5.0.

La documentation au format HTML des API de Java est fournie séparément. Malgré sa taille imposante, cette documentation est indispensable pour obtenir des informations complètes sur toutes les classes fournies. Le tableau ci dessous résume la taille des différents composants selon leur version pour la plate−forme Windows.

	Version 1.0	Version 1.1	Version 1.2	Version 1.3	Version 1.4	Version 5.0
JDK compressé		8,6 Mo	20 Mo	30 Mo	47 Mo	44 Mo
JDK installé				53 Mo	59 Mo	
JRE compressé			12 Mo	7 Mo		14 Mo
JRE installé				35 Mo	40 Mo	
Documentation compressée			16 Mo	21 Mo	30 Mo	43,5 Mo
Documentation décompressée			83 Mo	106 Mo	144 Mo	223 Mo

1.3.1. Java 1.0

Cette première version est lancée officiellement en mai 1995.

1.3.2. Java 1.1

Cette version du JDK est annoncée officiellement en mars 1997. Elle apporte de nombreuses améliorations et d'importantes fonctionnalités nouvelles dont :

- les Java beans
- les fichiers JAR
- RMI pour les objets distribués
- la sérialisation
- l'introspection
- JDBC pour l'accès aux données
- les classes internes

- l'internationalisation
- un nouveau modèle de sécurité permettant notamment de signer les applets
- JNI pour l'appel de méthodes natives
- ...

1.3.3. Java 1.2

Cette version du JDK est lancée fin 1998. Elle apporte de nombreuses améliorations et d'importantes fonctionnalités nouvelles dont :

- un nouveau modèle de sécurité basé sur les policy
- les JFC sont incluses dans le JDK (Swing, Java2D, accessibility, drag & drop ...)
- JDBC 2.0
- les collections
- support de CORBA
- un compilateur JIT est inclus dans le JDK
- de nouveaux format audio sont supportés
- ...

Java 2 se décline en 3 éditions différentes qui regroupent des APIs par domaine d'application :

- Java 2 Micro Edition (J2ME) : contient le nécessaire pour développer des applications capable de fonctionner dans des environnements limités tels que les assistants personnels (PDA), les téléphones portables ou les systèmes de navigation embarqués
- Java 2 Standard Edition (J2SE) : contient le nécessaire pour développer des applications et des applets. Cette édition reprend le JDK 1.0 et 1.1.
- Java 2 Enterprise Edition (J2EE) : contient un ensemble de plusieurs API permettant le développement d'applications destinées aux entreprises tel que JDBC pour l'accès aux bases de données, EJB pour développer des composants orientés métiers, Servlet / JSP pour générer des pages HTML dynamiques, ... Cette édition necessite le J2SE pour fonctionner.

Le but de ces trois éditions est de proposer une solution reposant sur Java quelque soit le type de développement à réaliser.

1.3.4. J2SE 1.3

Cette version du JDK apporte de nombreuses améliorations notamment sur les performances et des fonctionnalités nouvelles dont :

- JNDI est inclus dans le JDK
- hotspot est inclus dans la JVM
- ...

La rapidité d'exécution a été grandement améliorée dans cette version.

1.3.5. J2SE 1.4 (nom de code Merlin)

Cette version du JDK, lancée début 2002, apporte de nombreuses améliorations notamment sur les performances et des fonctionnalités nouvelles dont :

- JAXP est inclus dans le JDK pour le support de XML
- JDBC version 3.0
- new I/O API pour compléter la gestion des entrée/sortie
- logging API pour la gestion des logs applicatives
- une API pour utiliser les expressions régulières

- une api pour gérer les préférences utilisateurs
- JAAS est inclus dans le JDK pour l'authentification
- un ensemble d'API pour utiliser la cryptographie
- l'outil Java WebStart
- ...

Cette version ajoute un nouveau mot clé au langage pour utiliser les assertions : assert.

1.3.6. J2SE 5.0 (nom de code Tiger)

La version 1.5 du J2SE est spécifiée par le JCP sous la JSR 176. Elle devrait intégrer un certain nombre de JSR dans le but de simplifier les développements en Java.

Ces évolutions sont réparties dans une quinzaine de JSR qui seront intégrées dans la version 1.5 de Java.

JSR−003	JMX Management API
JSR−013	Decimal Arithmetic
JSR−014	Generic Types
JSR−028	SASL
JSR−114	JDBC API Rowsets
JSR−133	New Memory Model and thread
JSR−163	Profiling API
JSR−166	Concurrency Utilities
JSR−174	Monitoring and Management for the JVM
JSR−175	Metadata facility
JSR−199	Compiler APIs
JSR−200	Network Transfer Format for Java Archives
JSR−201	Four Language Updates
JSR−204	Unicode Surrogates
JSR−206	JAXP 1.3

La version 1.5 de Java est désignée officiellement sous le nom J2SE version 5.0.

1.3.7. Les futurs versions de Java

Version	Nom de code	Date de sortie
1.4	Merlin	2001
1.4.1	Hopper	2002
1.4.2	Mantis	2003
1.5	Tiger	2004
1.5.1	Dragonfly	???
1.6	Mustang	???

1.3.8. Le résumé des différentes versions

Au fur et à mesure des nouvelles versions de Java , le nombre de packages et de classes s'accroît :

	Java 1.0	Java 1.1	Java 1.2	J2SE 1.3	J2SE 1.4	J2SE 5.0
Nombre de de packages	8	23	59	76	135	166
Nombre de classes	201	503	1520	1840	2990	3280

1.3.9. Les extensions du JDK

Sun fourni un certains nombres d'API supplémentaires qui ne sont pas initialement fournies en standard dans le JDK. Ces API sont intégrées au fur et à mesure de l'évolution de Java.

Extension	Description
JNDI	Java Naming and directory interface Cet API permet d'unifier l'accès à des ressources. Elle est intégrée a Java 1.3
Java mail	Cette API permet de gérer des emails. Elle est intégrée a la plateforme J2EE.
Java 3D	Cette API permet de mettre en oeuvre des graphismes en 3 dimensions
Java Media	Cette API permet d'utiliser des composants multimédia
Java Servlets	Cette API permet de créer des servlets (composants serveurs). Elle est intégrée a la plateforme J2EE.
Java Help	Cette API permet de créer des aides en ligne pour les applications
Jini	Cette API permet d'utiliser Java avec des appareils qui ne sont pas des ordinateurs
JAXP	Cette API permet le parsing et le traitement de document XML. Elle est intégré a Java 1.4

Cette liste n'est pas exhaustive.

1.4. Un rapide tour d'horizon des API et de quelques outils

La communauté Java est très productive car elle regroupe :

- Sun, le fondateur de Java
- le JCP (Java Community Process) : c'est le processus de traitement des évolutions de Java dirigé par Sun. Chaque évolution est traitée dans une JSR (Java Specification Request) par un groupe de travail constitué de différents acteurs du monde Java
- des acteurs commerciaux dont tous les plus grands acteurs du monde informatique excepté Microsoft
- la communauté libre qui produit un très grand nombre d'API et d'outils pour Java

Ainsi l'ensemble des API et des outils utilisables est énorme et évolue très rapidement. Les tableaux ci dessous tentent de recenser les principaux par thème.

J2SE 1.4

Java Bean	RMI	IO	Applet
Reflexion	Collection	Logging	AWT
Net (réseau)	Preferences	Security	JFC
Internationalisation	Exp régulière		Swing

Les outils de Sun

Jar	Javadoc	Java Web Start	JWSDK

Les outils libres (les plus connus)

Jakarta Tomcat	Jakarta Ant	JBoss	Apache Axis
JUnit	Eclipse		

Les autres API

Données	Web	Entreprise	XML	Divers
JDBC	Servlets	Java Mail	JAXP	JAI
JDO	JSP	JNDI	SAX	JAAS
	JSTL	EJB	DOM	JCA
	Jave Server Faces	JMS	JAXM	JCE
		JMX	JAXR	Java Help
		JTA	JAX−RPC	JMF
		RMI−IIOP	SAAJ	JSSE
		Java IDL	JAXB	Java speech
		JINI		Java 3D
		JXTA		

Les API de la communauté open source

Données	Web	Entreprise	XML	Divers
OJB	Jakarta Struts		Apache Xerces	Jakarta Log4j
Castor	Webmacro		Apache Xalan	Jakarta regexp
Hibernate	Expresso		Apache Axis	
	Barracuda		JDOM	
	Turbine		DOM4J	

1.5. Les différences entre Java et JavaScript

Il ne faut pas confondre Java et JavaScript. JavaScript est un langage développé par Netscape Communications.

La syntaxe des deux langages est très proche car elles dérivent toutes les deux du C++.

Il existe de nombreuses differences entre les deux langages :

	Java	Javascript
Auteur	Développé par Sun Microsystems	Développé par Netscape Communications
Format	Compilé sous forme de byte−code	Interprété
Stockage	Applet téléchargé comme un élément de la page web	Source inséré dans la page web
Utilisation	Utilisable pour développer tous les types d'applications	Utilisable uniquement pour "dynamiser" les pages web
Exécution	Executé dans la JVM du navigateur	Executé par le navigateur
POO	Orienté objets	Manipule des objets mais ne permet pas d'en définir
Typage	Fortement typé	Pas de contrôle de type
Complexité du code	Code relativement complexe	Code simple

1.6. L'installation du JDK

Le JDK et la documentation sont librement téléchargeable sur le site web de Sun : http://java.sun.com

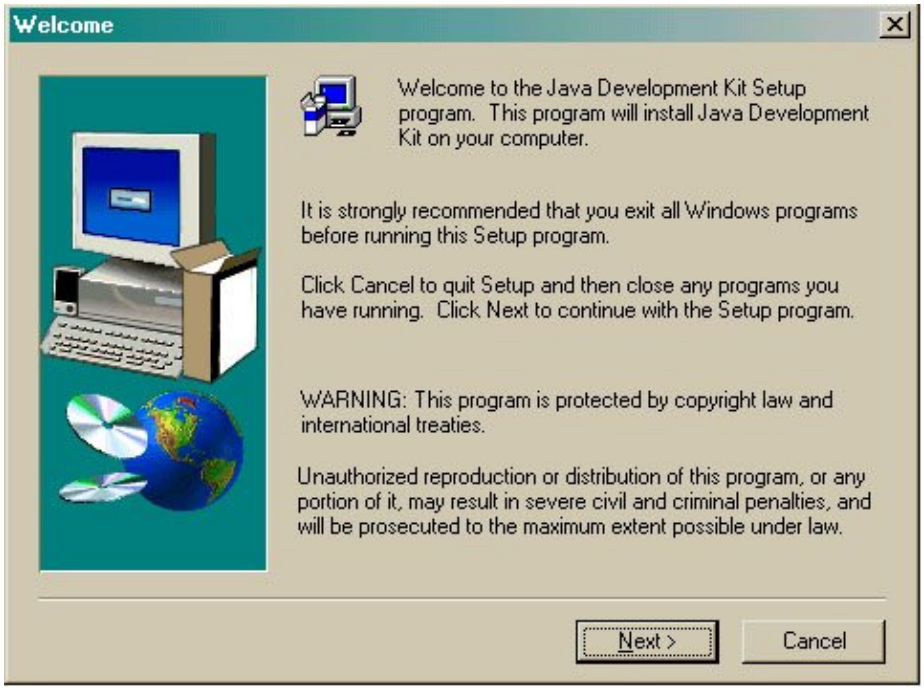
1.6.1. L'installation de la version 1.3 DU JDK de Sun sous Windows 9x

Pour installer le JDK 1.3 sous Windows 9x, il suffit de télécharger et d'exécuter le programme : j2sdk1_3_0−win.exe

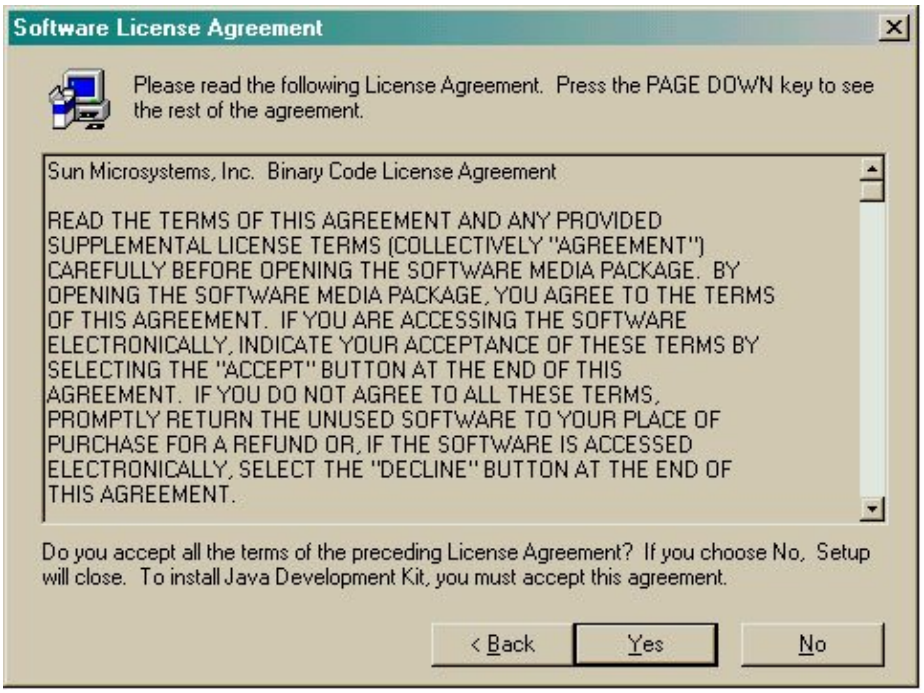
Le programme commence par désarchiver les composants.

Le programme utilise InstallShield pour guider et réaliser l'installation.

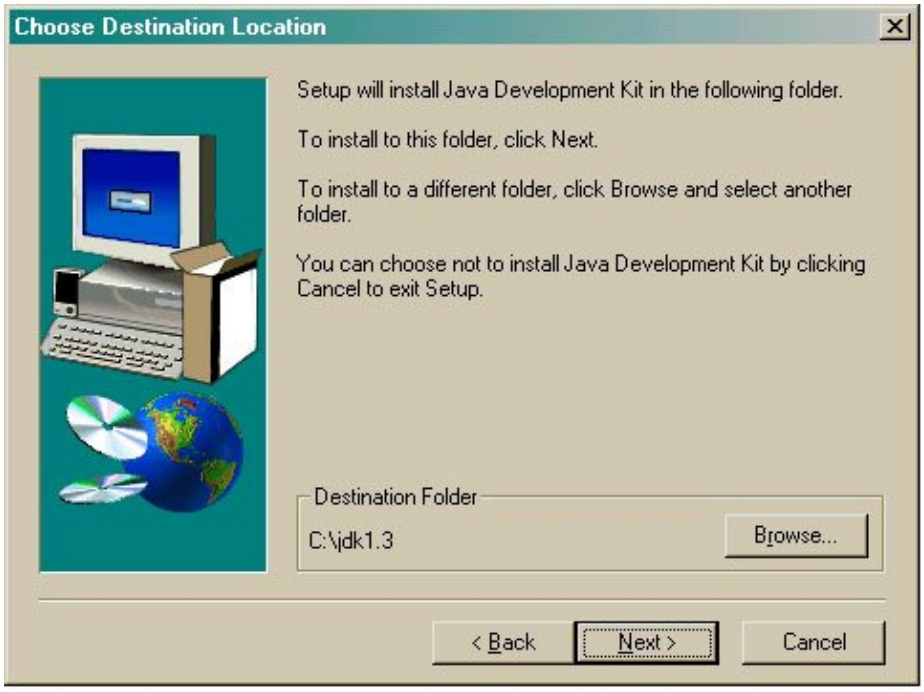
L'installation vous souhaite la bienvenue et vous donne quelques informations d'usage.



L'installation vous demande ensuite de lire et d'approuver les termes de la licence d'utilisation.

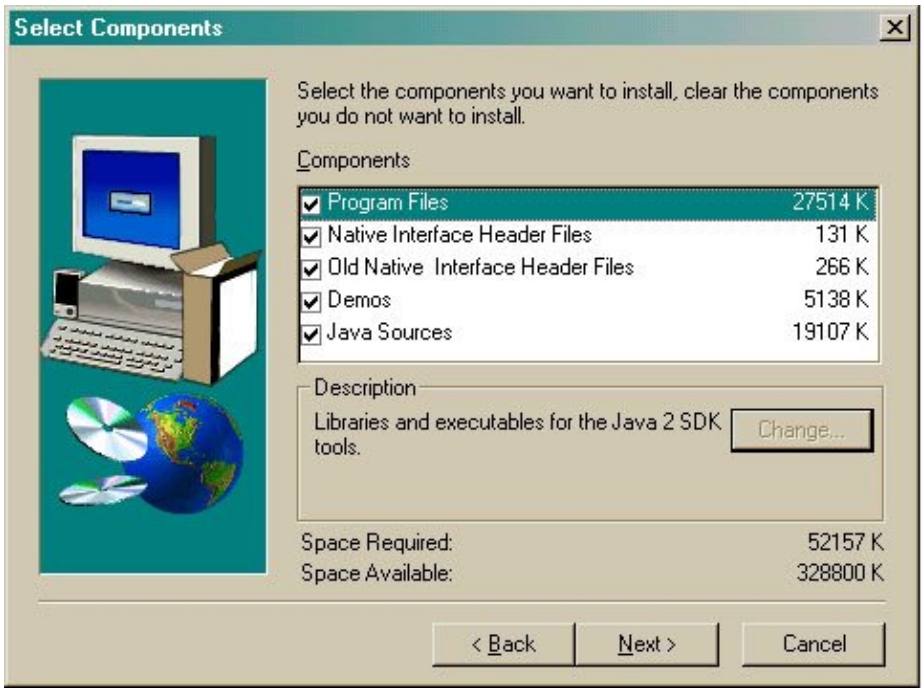


L'installation vous demande le répertoire dans lequel le JDK va être installé. Le répertoire proposé par défaut est pertinent car il est simple.



L'installation vous demande les composants à installer :

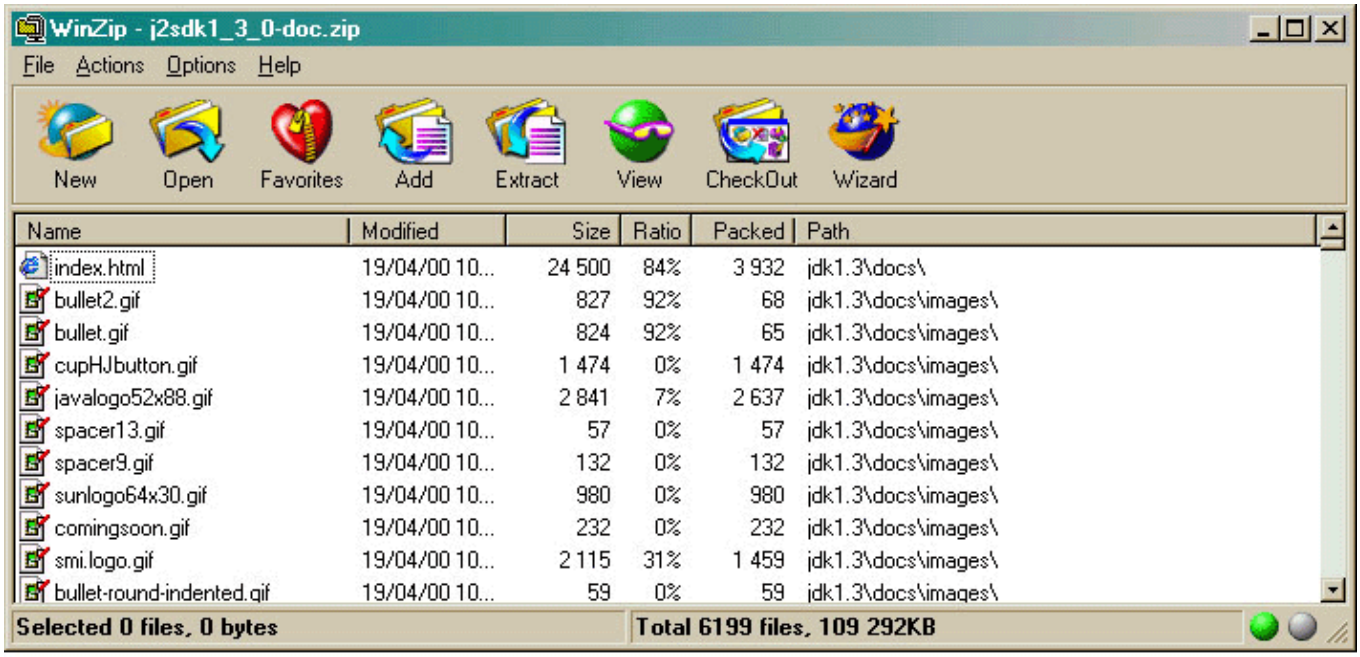
- Program Files est obligatoire pour une première installation
- Les interfaces natives ne sont utiles que pour réaliser des appels de code natif dans les programmes Java
- Les démos sont utiles car ils fournissent quelques exemples
- les sources contiennent les sources de la plupart des classes Java écrites en Java. Attention à l'espace disque nécessaire à cet élément



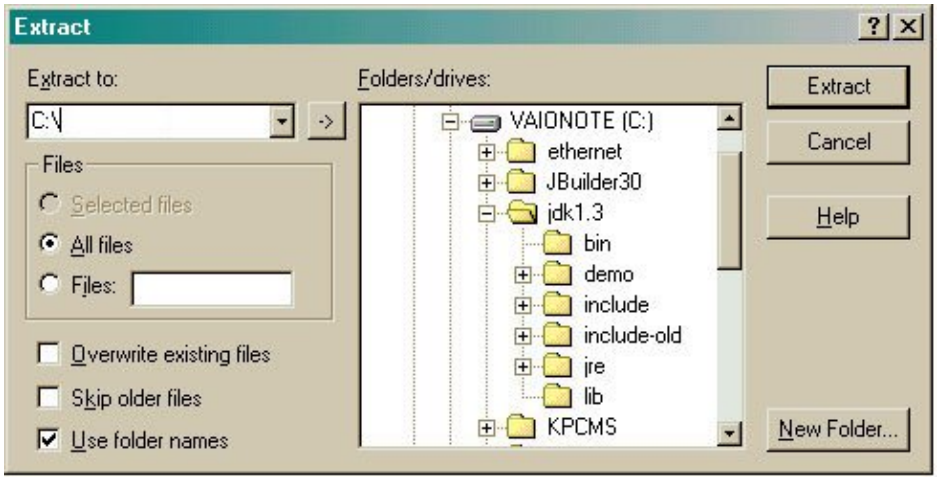
L'installation se poursuit par la copie des fichiers et la configuration du JRE.

1.6.2. L'installation de la documentation de Java 1.3 sous Windows

L'archive contient la documentation sous forme d'arborescence dont la racine est jdk1.3\docs.



Si le répertoire par défaut a été utilisé lors de l'installation, il suffit de décompresser l'archive à la racine du disque C:\.



Il peut être pratique de désarchiver le fichier dans un sous répertoire, ce qui permet de reunir plusieurs versions de la documentation.

1.6.3. La configuration des variables système sous Windows 9x

Pour un bon fonctionnement du JDK, il est recommandé de paramétrer correctement deux variables systèmes : la variable PATH qui définit les chemins de recherche des exécutables et la variable CLASSPATH qui définit les chemins de rechercher des classes et bibliothèques Java.

Pour configurer la variable PATH, il suffit d'ajouter à la fin du fichier autoexec.bat :

Exemple :
`SET PATH=%PATH%;C:\JDK1.3\BIN`

Attention : si une version antérieure du JDK était déjà présente, la variable PATH doit déjà contenir un chemin vers les utilitaires du JDK. Il faut alors modifier ce chemin sinon c'est l'ancienne version qui sera utilisée. Pour vérifier la version du JDK utilisée, il suffit de saisir la commande java −version dans une fenêtre DOS.

La variable CLASSPATH est aussi définie dans le fichier autoexec.bat. Il suffit d'ajouter une ligne ou de modifier la ligne existante définissant cette variable.

Exemple :

```
SET CLASSPATH=C:\JAVA\DEV;
```

Dans un environnement de développement, il est pratique d'ajouter le . qui désigne le répertoire courant dans le CLASSPATH surtout lorsque l'on n'utilise pas d'outils de type IDE. Attention toutefois, cette pratique est fortement déconseillée dans un environnement de production pour ne pas poser des problèmes de sécurité.

Il faudra ajouter par la suite les chemins d'accès aux différents packages requis par les développements afin de les faciliter.

Pour que ces modifications prennent effet dans le système, il faut redemarrer Windows ou exécuter ces deux instructions sur une ligne de commande DOS.

1.6.4. Les éléments du JDK 1.3 sous Windows

Le répertoire dans lequel a été installé le JDK contient plusieurs répertoires. Les répertoires donnés ci−après sont ceux utilisés en ayant gardé le répertoire par défaut lors de l'installation.

Répertoire	Contenu
C:\jdk1.3	Le répertoire d'installation contient deux fichiers intéressants : le fichier readme.html qui fourni quelques informations et des liens web et le fichier src.jar qui contient le source Jßava de nombreuses classes. Ce dernier fichier n'est présent que si l'option correspondante a été cochée lors de l'installation.
C:\jdk1.3\bin	Ce répertoire contient les exécutables : le compilateur javac, l'interpréteur java, le débuggeur jdb et d'une façon générale tous les outils du JDK.
C:\jdk1.3\demo	Ce répertoire n'est présent que si l'option nécessaire a été cochée lors de l'installation. Il contient des applications et des applets avec leur code source.
C:\jdk1.3\docs	Ce répertoire n'est présent que si la documentation a été décompressée.
C:\jdk1.3\include et C:\jdk1.3\include−old	Ces répertoires ne sont présents que si les options nécessaires ont été cochées lors de l'installation. Il contient des fichiers d'en−tête C (fichier avec l'extension .H) qui permettent de faire interagir du code Java avec du code natif
C:\jdk1.3\jre	Ce répertoire contient le JRE : il regroupe le nécessaire à l'exécution des applications notamment le fichier rt.jar qui regroupe les API. Depuis la version 1.3, le JRE contient deux machines virtuelles : la JVM classique et la JVM utilisant la technologie Hot spot. Cette dernière est bien plus rapide et c'est elle qui est utilisée par défaut. Les éléments qui composent le JRE sont séparés dans les répertoires bin et lib selon leur nature.
C:\jdk1.3\lib	Ce répertoire ne contient plus que quelques bibliothèques notamment le fichier tools.jar. Avec le JDK 1.1 ce répertoire contenait le fichier de la bibliothèque standard. Ce fichier est maintenant dans le répertoire JRE.

1.6.5. L'installation de la version 1.4.2 du JDK de Sun sous Windows

Télécharger sur le site java.sun.com et exécuter le fichier j2sdk−1_4_2_03−windows−i586−p.exe.

Un assistant permet de configurer l'installation au travers de plusieurs étapes :

- La page d'acceptation de la licence (« Licence agreement ») s'affiche
- Lire la licence et si vous l'acceptez, cliquer sur le bouton radio « I accept the terms in the licence agreement », puis cliquez sur le bouton « Next »
- La page de sélection des composants à installer (« Custom setup ») s'affiche, modifiez les composants à installer si nécessaire puis cliquez sur le bouton « Next »
- La page de sélection des plug in pour navigateur (« Browser registration ») permet de sélectionner les navigateurs pour lesquels le plug in Java sera installé, sélectionner ou non le ou les navigateurs détecté, puis cliquez sur le bouton « Install »
- L'installation s'opère en fonction des informations fournies précédemment
- La page de fin s'affiche, cliquez sur le bouton « Finish »

Même si ce n'est pas obligatoire pour fonctionner, il est particulièrement utile de configurer deux variables systèmes : PATH et CLASSPATH.

Dans la variable PATH, il est pratique de rajouter le chemin du répertoire bin du JDK installé pour éviter à chaque appel des commandes du JDK d'avoir à saisir leur chemin absolu.

Dans la variable CLASSPATH, il est pratique de rajouter les répertoires et les fichiers .jar qui peuvent être nécessaire lors des phases de compilation ou d'exécution, pour éviter d'avoir à les préciser à chaque fois.

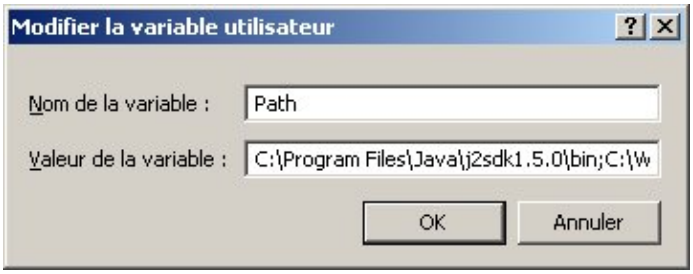
1.6.6. L'installation de la version 1.5 beta 1 du JDK de Sun sous Windows

Il faut télécharger sur le site de Sun et exécuter le fichier j2sdk−1_5_0−beta−windows−i586.exe

Un assistant guide l'utilisateur pour l'installation de l'outil.

- Sur la page « Licence Agreement », il faut lire la licence et si vous l'acceptez, cochez le bouton radio « I accept the terms in the licence agreement » et cliquez sur le bouton « Next »
- Sur la page « Custom Setup », il est possible de sélectionner/désélectionner les éléments à installer. Cliquez simplement sur le bouton « Next ».
- La page « Browser registration » permet de sélectionner les plug−ins des navigateurs qui seront installés. Cliquez sur le bouton « Install »
- Les fichiers sont copiés.
- La page « InstallShield Wizard Completed » s'affichage à la fin de l'installation. Cliquez sur « Finish ».

Pour faciliter l'utilisation des outils du J2SE SDK, il faut ajouter le chemin du répertoire bin contenant ces outils dans la variable Path du système.



Il est aussi utile de définir la variable d'environnement JAVA_HOME avec comme valeur le chemin d'installation du SDK.

1.6.7. Installation JDK 1.4.2 sous Linux Mandrake 10

La première chose est de décompresser le fichier téléchargé sur le site de Sun en exécutant le fichier dans un shell.

Exemple :

```
[java@localhost tmp]$ sh j2sdk-1_4_2_06-linux-i586-rpm.bin
               Sun Microsystems, Inc.
          Binary Code License Agreement
                    for the

JAVATM 2 SOFTWARE DEVELOPMENT KIT (J2SDK), STANDARD
EDITION, VERSION 1.4.2_X

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE
SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION
```

```
THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY
CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS
(COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT
...

Do you agree to the above license terms? [yes or no]
yes
Unpacking...
Checksumming...
0
0
Extracting...
UnZipSFX 5.40 of 28 November 1998, by Info-ZIP (Zip-Bugs@lists.wku.edu).
  inflating: j2sdk-1_4_2_06-linux-i586.rpm
Done.
[java@localhost tmp]$
```

Le décompression créé un fichier j2sdk-1_4_2_06-linux-i586.rpm. Pour installer ce package, il est nécessaire d'être root sinon son installation est impossible.

Exemple :

```
[java@localhost eclipse3]$ rpm -ivh j2sdk-1_4_2_06-linux-i586.rpm
erreur: cannot open lock file ///var/lib/rpm/RPMLOCK in exclusive mode
erreur: impossible d'ouvrir la base de données Package dans /var/lib/rpm

[java@localhost eclipse3]$ su root
Password:
[root@localhost eclipse3]# rpm -ivh j2sdk-1_4_2_06-linux-i586.rpm
Préparation...              ########################################### [100%]
   1:j2sdk                  ########################################### [100%]
[root@localhost eclipse3]#
```

Le JDK a été installé dans le répertoire /usr/java/j2sdk1.4.2_06

Pour permettre l'utilisation par tous les utilisateurs du système, le plus simple est de créer un fichier de configuration dans le répertoire /etc/profile.d

Créez un fichier java.sh

Exemple : le contneu du fichier java.sh

```
[root@localhost root]# cat java.sh
export JAVA_HOME="/usr/java/j2sdk1.4.2_06"
export PATH=$PATH:$JAVA_HOME/bin
```

Modifiez ces droits pour permettre son éxecution

Exemple :

```
[root@localhost root]# chmod 777 java.sh
[root@localhost root]# source java.sh
```

Si kaffe est déjà installé sur le système il est préférable de mettre le chemin vers le JDK en tête de la variable PATH

Exemple :

```
[root@localhost root]# java
usage: kaffe [-options] class
Options are:
        -help                   Print this message
        -version                Print version number
        -fullversion            Print verbose version info
```

```
        -ss <size>              Maximum native stack size
[root@localhost root]# cat java.sh
export JAVA_HOME="/usr/java/j2sdk1.4.2_06"
export PATH=$JAVA_HOME/bin:$PATH
```

Pour rendre cette modification permanente, il faut copier le fichier java.sh dans le répertoire /etc/profile.d

Exemple :

```
[root@localhost root]# cp java.sh /etc/profile.d
```

Ainsi tous utilisateurs qui ouvriront une nouvelle console Bash aura ces variables d'environnements positionnées pour utiliser les outils du JDK.

Exemple :

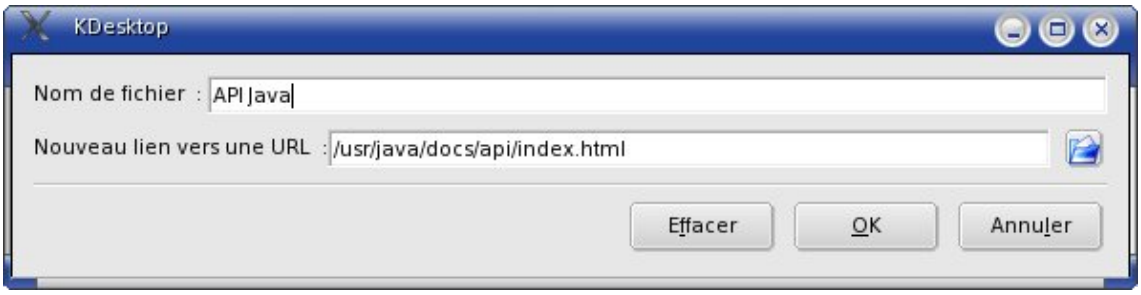
```
[java@localhost java]$ echo $JAVA_HOME
/usr/java/j2sdk1.4.2_06
[java@localhost java]$ java -version
java version "1.4.2_06"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_06-b03)
Java HotSpot(TM) Client VM (build 1.4.2_06-b03, mixed mode)
[java@localhost java]$
```

L'installation de la documentation se en décompressant l'archive dans un répertoire du système par exemple /usr/java.

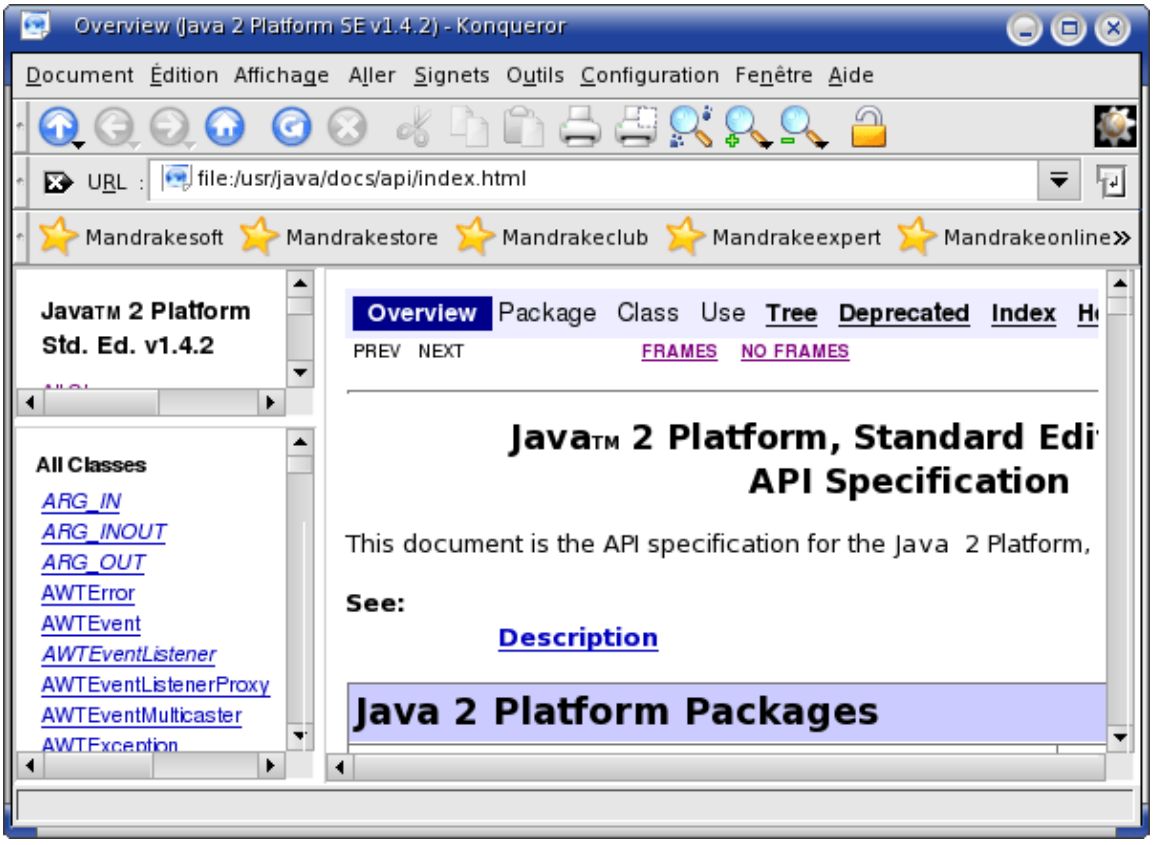
Exemple :

```
[root@localhost local]# mv j2sdk-1_4_2-doc.zip /usr/java
[root@localhost java]# ll
total 33636
drwxr-xr-x  8 root root     4096 oct 16 22:18 j2sdk1.4.2_06/
-rwxr--r--  1 root root 34397778 oct 18 23:39 j2sdk-1_4_2-doc.zip*
[root@localhost java]# unzip -q j2sdk-1_4_2-doc.zip
[root@localhost java]# ll
total 33640
drwxrwxr-x  8 root root     4096 aoû 15  2003 docs/
drwxr-xr-x  8 root root     4096 oct 16 22:18 j2sdk1.4.2_06/
-rwxr--r--  1 root root 34397778 oct 18 23:39 j2sdk-1_4_2-doc.zip*
[root@localhost java]# rm j2sdk-1_4_2-doc.zip
rm: détruire fichier régulier `j2sdk-1_4_2-doc.zip'? o
[root@localhost java]# ll
total 8
drwxrwxr-x  8 root root 4096 aoû 15  2003 docs/
drwxr-xr-x  8 root root 4096 oct 16 22:18 j2sdk1.4.2_06/
[root@localhost java]#
```

Il est possible pour un utilisateur de créer un raccourci sur le bureau KDE en utilisant le menu contextuel créer un « nouveau/fichier/lien vers une url ... »



Un double clic sur la nouvelle icône permet d'ouvrir directement Konqueror avec l'aide en ligne de l'API.


Overview (Java 2 Platform SE v1.4.2) - Konqueror
Document Édition Affichage Aller Signets Outils Configuration Fenêtre Aide
URL : file:/usr/java/docs/api/index.html
Mandrakesoft Mandrakestore Mandrakeclub Mandrakeexpert Mandrakeonline
Java™ 2 Platform
Std. Ed. v1.4.2
All Classes
ARG_IN
ARG_INOUT
ARG_OUT
AWTError
AWTEvent
AWTEventListener
AWTEventListenerProxy
AWTEventMulticaster
AWTException
Overview Package Class Use Tree Deprecated Index
PREV NEXT
FRAMES NO FRAMES
Java™ 2 Platform, Standard Edi
API Specification
This document is the API specification for the Java 2 Platform,
See:
Description
Java 2 Platform Packages

2. Les techniques de base de programmation en Java

N'importe quel éditeur de texte peut être utilisé pour éditer un fichier source Java.

Il est nécessaire de compiler le source pour le transformer en J−code ou byte−code Java qui sera lui exécuté par la machine virtuelle.

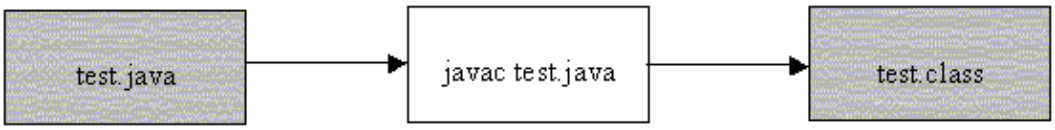
Il est préférable de définir une classe par fichier. Le nom de la classe publique et le fichier qui la contient doivent être identiques.

Pour être compilé, le programme doit être enregistré au format de caractères Unicode : une conversion automatique est faite par le JDK si nécessaire.

Ce chapitre contient plusieurs sections :

- La compilation d'un code source : cette section présente la compilation d'un fichier source.
- L'exécution d'un programme et d'une applet : cette section présente l'exécution d'un programme et d'une applet.

2.1. La compilation d'un code source



Pour compiler un fichier source il suffit d'invoquer la commande javac avec le nom du fichier source avec son extension .java

```
javac NomFichier.java
```

Le nom du fichier doit correspondre au nom de la classe principale en respectant la casse même si le système d'exploitation n'y est pas sensible

Suite à la compilation, le pseudo code Java est enregistré sous le nom NomFichier.class

2.2. L'exécution d'un programme et d'une applet

2.2.1. L'exécution d'un programme

Une classe ne peut être exécutée que si elle contient une méthode main() correctement définie.

Pour exécuter un fichier contenant du byte−code il suffit d'invoquer la commande java avec le nom du fichier source sans son extension .class

```
java NomFichier
```

2.2.2. L'exécution d'une applet

Il suffit de créer une page HTML pouvant être très simple :

Exemple

```
<HTML>
<TITLE> test applet Java </TITLE>
<BODY>
<APPLET code=« NomFichier.class » width=270 height=200>
</APPLET>
</BODY>
</HTML>
```

Il faut ensuite visualiser la page créée dans l'appletviewer ou dans un navigateur 32 bits compatible avec la version de Java dans laquelle l'applet est écrite.

3. La syntaxe et les éléments de bases de Java

Ce chapitre contient plusieurs sections :

- Les règles de base : cette section présente les règles syntaxiques de base de Java.
- Les identificateurs : cette section présente les règles de composition des identificateurs.
- Les commentaires : cette section présente les différentes formes de commentaires de Java.
- La déclaration et l'utilisation de variables : cette section présente la déclaration des variables, les types élémentaires, les formats des type élémentaires, l'initialisation des variables, l'affectation et les comparaisons.
- Les opérations arithmétiques : cette section présente les opérateurs arithmétique sur les entiers et les flottants et les opérateurs d'incrémentation et de décrémentation.
- La priorité des opérateurs : cette section présente la priorité des opérateurs.
- Les structures de contrôles : cette section présente les instructions permettant la réalisation de boucles, de branchements conditionnels et de débranchements.
- Les tableaux : cette section présente la déclaration, l'initialisation explicite et le parcours d'un tableau
- Les conversions de types : cette section présente la conversion de types élémentaires.
- La manipulation des chaînes de caractères : cette section présente la définition et la manipulation de chaîne de caractères (addition, comparaison, changement de la casse ...).

3.1. Les règles de base

Java est sensible à la casse.

Les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ';' (point virgule).

Une instruction peut tenir sur plusieurs lignes :

Exemple :

```
char
code
=
'D';
```

L'indentation est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur.

3.2. Les identificateurs

Chaque objet, classe, programme ou variable est associé à un nom : l'identificateur qui peut se composer de tous les caractères alphanumériques et des caractères _ et $. Le premier caractère doit être une lettre, le caractère de soulignement ou le signe dollar.

Rappel : Java est sensible à la casse.

Un identificateur ne peut pas appartenir à la liste des mots réservé du langage Java :

abstract	assert (JDK 1.4)	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	false	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	null	package	private	protected
public	return	short	static	super
switch	synchronized	this	throw	throws
transient	true	try	void	volatile
while				

3.3. Les commentaires

Ils ne sont pas pris en compte par le compilateur donc ils ne sont pas inclus dans le pseudo code. Ils ne se terminent pas par un ;.

Il existe trois types de commentaire en Java :

Type de commentaires	Exemple
commentaire abrégé	`// commentaire sur une seule ligne` `int N=1; // déclaration du compteur`
commentaire multiligne	`/* commentaires ligne 1` `commentaires ligne 2 */`

commentaire de documentation automatique	`/**`
	`  * commentaire de la methode`
	`  * @param val la valeur a traiter`
	`  * @since 1.0`
	`  * @return Rien`
	`  * @deprecated Utiliser la nouvelle methode XXX`
	`  */`

3.4. La déclaration et l'utilisation de variables

3.4.1. La déclaration de variables

Une variable possède un nom, un type et une valeur. La déclaration d'une variable doit donc contenir deux choses : un nom et le type de données qu'elle peut contenir. Une variable est utilisable dans le bloc ou elle est définie.

La déclaration d'une variable permet de réserver la mémoire pour en stocker la valeur.

Le type d'une variable peut être :

- soit un type élémentaire dit aussi type primitif déclaré sous la forme **type_élémentaire variable;**
- soit une classe déclarée sous la forme **classe variable ;**

Exemple :

```
long nombre;
int compteur;
String chaine;
```

Rappel : les noms de variables en Java peuvent commencer par un lettre, par le caractère de soulignement ou par le signe dollar. Le reste du nom peut comporter des lettres ou des nombres mais jamais d'espaces.

Il est possible de définir plusieurs variables de même type en séparant chacune d'elles par une virgule.

Exemple :

```
int jour, mois, annee ;
```

Java est un langage à typage rigoureux qui ne possède pas de transtypage automatique lorsque ce transtypage risque de conduire à une perte d'information.

Pour les objets, il est nécessaire en plus de la déclaration de la variable de créer un objet avant de pouvoir l'utiliser. Il faut réserver de la mémoire pour la création d'un objet (remarque : un tableau est un objet en Java) avec l'instruction **new**. La libération de la mémoire se fait automatiquement grâce au garbage collector.

Exemple :

```
MaClasse instance; // déclaration de l'objet

instance = new MaClasse(); // création de l'objet

OU MaClasse instance = new MaClasse(); // déclaration et création de l'objet
```

Exemple :

```
int[] nombre = new int[10];
```

Il est possible en une seule instruction de faire la déclaration et l'affectation d'une valeur à une variable ou plusieurs variables.

Exemple :

```
int i=3 , j=4 ;
```

3.4.2. Les types élémentaires

Les types élémentaires ont une taille identique quelque soit la plate−forme d'exécution : c'est un des éléments qui permet à Java d'être indépendant de la plate−forme sur lequel le code s'exécute.

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique : true ou false	1 bit	true ou false	pas de conversion possible vers un autre type
byte	octet signé	8 bits	−128 à 127	
short	entier court signé	16 bits	−32768 à 32767	
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans du code Java
int	entier signé	32 bits	−2147483648 à 2147483647	

float	virgule flottante simple précision (IEEE754)	32 bits	1.401e−045 à 3.40282e+038	
double	virgule flottante double précision (IEEE754)	64 bits	2.22507e−308 à 1.79769e+308	
long	entier long	64 bits	−9223372036854775808 à 9223372036854775807	

Les types élémentaires commencent tous par une minuscule.

3.4.3. Le format des types élémentaires

Le format des nombres entiers :

Les types byte, short, int et long peuvent être codés en décimal, hexadécimal ou octal. Pour un nombre hexadécimal, il suffit de préfixer sa valeur par 0x. Pour un nombre octal, le nombre doit commencer par un zéro. Le suffixe l ou L permet de spécifier que c'est un entier long.

Le format des nombres décimaux :

Les types float et double stockent des nombres flottants : pour être reconnus comme tel ils doivent posséder soit un point, un exposant ou l'un des suffixes f, F, d, D. Il est possible de préciser des nombres qui n'ont pas le partie entière ou décimale.

Exemple :

```
float pi = 3.141f;
double v = 3d
float f = +.1f , d = 1e10f;
```

Par défaut un littéral est de type double : pour définir un float il faut le suffixer par la lettre f ou F.

Exemple :

```
double w = 1.1;
```

Attention : `float pi = 3.141; // erreur à la compilation`

Le format des caractères :

Un caractère est codé sur 16 bis car il est conforme à la norme Unicode. Il doit être entouré par des apostrophes. Une valeur de type char peut être considérée comme un entier non négatif de 0 à 65535. Cependant la conversion implicite par affectation n'est pas possible.

Exemple :

```
/* test sur les caractères */
class test1 {
    public static void main (String args[]) {
        char code = 'D';
        int index = code - 'A';
        System.out.println("index = " + index);
    }
}
```

3.4.4. L'initialisation des variables

Exemple :

```
int nombre; // déclaration
nombre = 100; //initialisation
OU int nombre = 100; //déclaration et initialisation
```

En Java, toute variable appartenant à un objet (définie comme étant un attribut de l'objet) est initialisée avec une valeur par défaut en accord avec son type au moment de la creation. Cette initialisation ne s'applique pas aux variables locales des méthodes de la classe.

Les valeurs par défaut lors de l'initialisation automatique des variables d'instances sont :

Type	Valeur par défaut
boolean	false
byte, short, int, long	0
float, double	0.0
char	\u000
classe	null

Remarque : Dans une applet, il est préférable de faire les déclarations et initialisation dans la méthode init().

3.4.5. L'affectation

le signe = est l'opérateur d'affectation et s'utilise avec une expression de la forme variable = expression. L'opération d'affectation est associatif de droite à gauche : il renvoie la valeur affectée ce qui permet d'écrire :

```
x = y = z = 0;
```

Il existe des opérateurs qui permettent de simplifier l'écriture d'une opération d'affectation associée à un opérateur mathématique :

Opérateur	Exemple	Signification
=	a=10	équivalent à : a = 10
+=	a+=10	équivalent à : a = a + 10
−=	a−=	équivalent à : a = a − 10
*=	a*=	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	a%=10	reste de la division
^=	a^=10	équivalent à : a = a ^ 10
<<=	a<<=10	équivalent à : a = a << 10 a est complété par des zéros à droite
>>=	a>>=10	équivalent à : a = a >> 10 a est complété par des zéros à gauche
>>>=	a>>>=10	équivalent à : a = a >>> 10 décalage à gauche non signé

Attention : Lors d'une opération sur des opérandes de types différents, le compilateur détermine le type du résultat en prenant le type le plus précis des opérandes. Par exemple, une multiplication d'une variable de type float avec une variable de type double donne un résultat de type double. Lors d'une opération entre un opérande entier et un flottant, le résultat est du type de l'opérande flottant.

3.4.6. Les comparaisons

Java propose des opérateurs pour toutes les comparaisons :

Opérateur	Exemple	Signification
`>`	`a > 10`	strictement supérieur
`<`	`a < 10`	strictement inférieur
`>=`	`a >= 10`	supérieur ou égal
`<=`	`a <= 10`	inférieur ou égal
`==`	`a == 10`	Egalité
`!=`	`a != 10`	diffèrent de
`&`	`a & b`	ET binaire
`^`	`a ^ b`	OU exclusif binaire
`|`	`a | b`	OU binaire
`&&`	`a && b`	ET logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient fausse
`||`	`a || b`	OU logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient vraie
`? :`	`a ? b : c`	opérateur conditionnel : renvoie la valeur b ou c selon l'évaluation de l'expression a (si a alors b sinon c) : b et c doivent retourner le même type

Les opérateurs sont exécutés dans l'ordre suivant à l'intérieure d'une expression qui est analysée de gauche à droite:

- incréments et décréments
- multiplication, division et reste de division (modulo)
- addition et soustraction
- comparaison
- le signe = d'affectation d'une valeur à une variable

L'usage des parenthèses permet de modifier cet ordre de priorité.

3.5. Les opérations arithmétiques

Les opérateurs arithmétiques se notent + (addition), − (soustraction), * (multiplication), / (division) et % (reste de la division). Ils peuvent se combiner à l'opérateur d'affectation

Exemple :
`nombre += 10;`

3.5.1. L'arithmétique entière

Pour les types numériques entiers, Java met en oeuvre une sorte de mécanisme de conversion implicite vers le type int appelée promotion entière. Ce mécanisme fait partie des règles mise en place pour renforcer la sécurité du code.

Exemple :

```
short x= 5 , y = 15;
x = x + y ; //erreur à la compilation

Incompatible type for =. Explicit cast needed to convert int to short.
x = x + y ; //erreur à la compilation
^
1 error
```

Les opérandes et le résultat de l'opération sont convertis en type int. Le résultat est affecté dans un type short : il y a donc risque de perte d'informations et donc erreur à la compilation est émise. Cette promotion évite un débordement de capacité sans que le programmeur soit pleinement conscient du risque : il est nécessaire, pour régler le problème, d'utiliser une conversion explicite ou cast

Exemple :

```
x = (short) ( x + y );
```

Il est nécessaire de mettre l'opération entre parenthèse pour que ce soit son résultat qui soit converti car le cast a une priorité plus forte que les opérateurs arithmétiques.

La division par zéro pour les types entiers lève l'exception ArithmeticException

Exemple :

```
/* test sur la division par zero de nombres entiers */
class test3 {
    public static void main (String args[]) {
        int valeur=10;
        double résultat = valeur / 0;
        System.out.println("index = " + résultat);
    }
}
```

3.5.2. L'arithmétique en virgule flottante

Avec des valeurs float ou double, la division par zéro ne produit pas d'exception mais le résultat est indiqué par une valeur spéciale qui peut prendre trois états :

- indéfini : Float.NaN ou Double.NaN (not a number)
- indéfini positif : Float.POSITIVE_INFINITY ou Double.POSITIVE_INFINITY, + ∞
- indéfini négatif : Float.NEGATIVE_INFINITY ou Double.NEGATIVE_INFINITY, + ∞

Conformément à la norme IEEE754, ces valeurs spéciales représentent le résultat d'une expression invalide NaN, une valeur supérieure au plafond du type pour infini positif ou négatif.

X	Y	X / Y	X % Y
valeur finie	0	+ ∞	NaN
valeur finie	+/− ∞	0	x
0	0	NaN	NaN
+/− ∞	valeur finie	+/− ∞	NaN
+/− ∞	+/− ∞	NaN	NaN

Exemple :

```
/* test sur la division par zero de nombres flottants */

class test2 {
    public static void main (String args[]) {
        float valeur=10f;
        double résultat = valeur / 0;
        System.out.println("index = " + résultat);
    }
}
```

3.5.3. L'incrémentation et la décrémentation

Les opérateurs d'incrémentation et de décrémentation sont : `n++ ++n n-- --n`

Si l'opérateur est placé avant la variable (préfixé), la modification de la valeur est immédiate sinon la modification n'a lieu qu'à l'issu de l'exécution de la ligne d'instruction (postfixé)

L'opérateur ++ renvoie la valeur avant incrémentation s'il est postfixé, après incrémentation s'il est préfixé.

Exemple :

```
System.out.println(x++); // est équivalent à
System.out.println(x); x = x + 1;

System.out.println(++x); // est équivalent à
x = x + 1; System.out.println(x);
```

Exemple :

```
/* test sur les incrementations prefixees et postfixees */

class test4 {
    public static void main (String args[]) {
        int n1=0;
        int n2=0;
        System.out.println("n1 = " + n1 + " n2 = " + n2);
        n1=n2++;
        System.out.println("n1 = " + n1 + " n2 = " + n2);
        n1=++n2;
        System.out.println("n1 = " + n1 + " n2 = " + n2);
        n1=n1++;        //attention
        System.out.println("n1 = " + n1 + " n2 = " + n2);
    }
}
```

Résultat :

```
int n1=0;

int n2=0; // n1=0 n2=0

n1=n2++; // n1=0 n2=1

n1=++n2; // n1=2 n2=2

n1=n1++; // attention : n1 ne change pas de valeur
```

3.6. La priorité des opérateurs

Java définit les priorités dans les opérateurs comme suit (du plus prioritaire au moins prioritaire)

les parenthèses	( )

les opérateurs d'incrémentation	++ −−
les opérateurs de multiplication, division, et modulo	* / %
les opérateurs d'addition et soustraction	+ −
les opérateurs de décalage	<< >>
les opérateurs de comparaison	< > <= >=
les opérateurs d'égalité	== !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	|
l'opérateur ET logique	&&
l'opérateur OU logique	||
les opérateurs d'assignement	= += −=

Les parenthèses ayant une forte priorité, l'ordre d'interprétation des opérateurs peut être modifié par des parenthèses.

3.7. Les structures de contrôles

Comme quasi totalité des langages de développement orienté objets, Java propose un ensemble d'instructions qui permettent de d'organiser et de structurer les traitements. L'usage de ces instructions est similaire à celui rencontré dans leur équivalent dans d'autres langages.

3.7.1. Les boucles

```
while ( boolean )
{
    ... // code a exécuter dans la boucle
}
```

Le code est exécuté tant que le booléen est vrai. Si avant l'instruction while, le booléen est faux, alors le code de la boucle ne sera jamais exécuté

Ne pas mettre de ; après la condition sinon le corps de la boucle ne sera jamais exécuté

```
do {
   ...
} while ( boolean )
```

Cette boucle est au moins exécuté une fois quelque soit la valeur du booléen;

```
for ( initialisation; condition; modification) {
    ...
}
```

Exemple :

```
for (i = 0 ; i < 10; i++ ) { ....}
for (int i = 0 ; i < 10; i++ ) { ....}
for ( ; ; ) { ... } // boucle infinie
```

L'initialisation, la condition et la modification de l'index sont optionels.

Dans l'initialisation, on peut déclarer une variable qui servira d'index et qui sera dans ce cas locale à la boucle.

Il est possible d'inclure plusieurs traitements dans l'initialisation et la modification de la boucle : chacun des traitements doit être séparé par une virgule.

Exemple :

```
for (i = 0 , j = 0 ; i * j < 1000;i++ , j+= 2) { ....}
```

La condition peut ne pas porter sur l'index de la boucle :

Exemple :

```
boolean trouve = false;
for (int i = 0 ; !trouve ; i++ ) {
    if ( tableau[i] == 1 )
    trouve = true;
    ... //gestion de la fin du parcours du tableau
}
```

Il est possible de nommer une boucle pour permettre de l'interrompre même si cela est peu recommendé :

Exemple :

```
int compteur = 0;
boucle:
while (compteur < 100) {

    for(int compte = 0 ; compte < 10 ; compte ++) {
        compteur += compte;
        System.out.println("compteur = "+compteur);
        if (compteur> 40) break boucle;
    }
}
```

3.7.2. Les branchements conditionnels

```
if (boolean) {
    ...
} else if (boolean) {
    ...
} else {
    ...
}
```

```
switch (expression) {
    case constante1 :
        instr11;
        instr12;
        break;

    case constante2 :
        ...
    default :
        ...
}
```

On ne peut utiliser switch qu'avec des types primitifs d'une taille maximum de 32 bits (byte, short, int, char).

Si une instruction case ne contient pas de break alors les traitements associés au case suivant sont exécutés.

Il est possible d'imbriquer des switch

L'opérateur ternaire : (condition) **?** valeur−vrai **:** valeur−faux

Exemple :

```
if (niveau == 5) // equivalent à total = (niveau ==5) ? 10 : 5;
total = 10;
else total = 5 ;
System.out.println((sexe == « H ») ? « Mr » : « Mme »);
```

3.7.3. Les débranchements

break : permet de quitter immédiatement une boucle ou un branchement. Utilisable dans tous les contrôles de flot

continue : s'utilise dans une boucle pour passer directement à l'itération suivante

break et continue peuvent s'excuter avec des blocs nommés. Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break.

Une étiquette est un nom suivi d'un deux points qui définit le début d'une instruction.

3.8. Les tableaux

Ils sont dérivés de la classe Object : il faut utiliser des méthodes pour y accéder dont font partie des messages de la classe Object tel que equals() ou getClass().

Le premier élément d'un tableau possède l'indice 0.

3.8.1. La déclaration des tableaux

Java permet de placer les crochets après ou avant le nom du tableau dans la déclaration.

Exemple :

```
int tableau[] = new int[50]; // déclaration et allocation

OU int[] tableau = new int[50];

OU int tab[]; // déclaration
tab = new int[50]; //allocation
```

Java ne supporte pas directement les tableaux à plusieurs dimensions : il faut déclarer un tableau de tableau.

Exemple :

```
float tableau[][] = new float[10][10];
```

La taille des tableaux de la seconde dimension peut ne pas être identique pour chaque occurrence.

Exemple :

```
int dim1[][] = new int[3][];
dim1[0] = new int[4];
dim1[1] = new int[9];
dim1[2] = new int[2];
```

Chaque élément du tableau est initialisé selon son type par l'instruction new : 0 pour les numériques, '\0' pour les caractères, false pour les booléens et nil pour les chaines de caractères et les autres objets.

3.8.2. L'initialisation explicite d'un tableau

Exemple :

```
int tableau[5] = {10,20,30,40,50};
int tableau[3][2] = {{5,1},{6,2},{7,3}};
```

La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création.

Exemple :

```
int tableau[] = {10,20,30,40,50};
```

Le nombre d'élément de chaque lignes peut ne pas être identique :

Exemple :

```
int[][] tabEntiers = {{1,2,3,4,5,6},
                      {1,2,3,4},
                      {1,2,3,4,5,6,7,8,9}};
```

3.8.3. Le parcours d'un tableau

Exemple :

```
for (int i = 0; i < tableau.length ; i ++) { ... }
```

La variable **length** retourne le nombre d'éléments du tableau.

Pour passer un tableau à une méthode, il suffit de déclarer les paramètres dans l'en tête de la méthode

Exemple :

```
public void printArray(String texte[]){ ...
}
```

Les tableaux sont toujours transmis par référence puisque ce sont des objets.

Un accès a un élément d'un tableau qui dépasse sa capacité, lève une exception du type java.lang.arrayIndexOutOfBoundsException.

3.9. Les conversions de types

Lors de la déclaration, il est possible d'utiliser un cast :

Exemple :

```
int entier = 5;
float flottant = (float) entier;
```

La conversion peut entrainer une perte d'informations.

Il n'existe pas en Java de fonction pour convertir : les conversions de type se font par des méthodes. La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion de types élémentaires.

Classe	Role
String	pour les chaines de caractères Unicode
Integer	pour les valeurs entières (integer)
Long	pour les entiers long signés (long)
Float	pour les nombres à virgules flottante (float)
Double	pour les nombres à virgule flottante en double précision (double)

Les classes portent le même nom que le type élémentaire sur lequel elles reposent avec la première lettre en majuscule.

Ces classes contiennent généralement plusieurs constructeurs. Pour y accéder, il faut les instancier puisque ce sont des objets.

Exemple :

```
String montexte;
 montexte = new String(«test»);
```

L'objet montexte permet d'accéder aux méthodes de la classe java.lang.String

3.9.1. La conversion d'un entier int en chaîne de caractère String

Exemple :

```
int i = 10;
String montexte = new String();
montexte =montexte.valueOf(i);
```

valueOf est également définie pour des arguments de type boolean, long, float, double et char

3.9.2. La conversion d'une chaîne de caractères String en entier int

Exemple :

```
String montexte = new String(« 10 »);
Integer monnombre=new Integer(montexte);
int i = monnombre.intValue(); //conversion d'Integer en int
```

3.9.3. La conversion d'un entier int en entier long

Exemple :

```
int i=10;
Integer monnombre=new Integer(i);
long j=monnombre.longValue();
```

3.10. La manipulation des chaînes de caractères

La définition d'un caractère se fait grâce au type char :

Exemple :

```
char touche = '%';
```

La définition d'une chaîne se fait grâce à l'objet String :

Exemple :

```
String texte = « bonjour »;
```

Les variables de type String sont des objets. Partout ou des constantes chaînes de caractères figurent entre guillemets, le compilateur Java génère un objet de type String avec le contenu spécifié. Il est donc possible d'ecrire :

```
String texte = « Java Java Java ».replace('a','o');
```

Les chaînes de caractères ne sont pas des tableaux : il faut utiliser les méthodes de la classe String d'un objet instancié pour effectuer des manipulations.

Il est impossible de modifier le contenu d'un objet String contruit à partir d'une constante. Cependant, il est possible d'utiliser les méthodes qui renvoient une chaîne pour modifier le contenu de la chaîne.

Exemple :

```
String texte = « Java Java Java »;
texte = texte.replace('a','o');
```

Java ne fonctionne pas avec le jeu de caractères ASCII ou ANSI, mais avec Unicode (Universal Code). Ceci concerne les types char et les chaînes de caractères. Le jeu de caractères Unicode code un caractère sur 2 octets. Les caractères 0 à 255 correspondent exactement au jeu de caractères ASCII étendu.

3.10.1. Les caractères spéciaux dans les chaines

Caractères spéciaux	Affichage
\'	Apostrophe
\ »	Guillemet
\\	anti slash
\t	Tabulation
\b	retour arrière (backspace)
\r	retour chariot
\f	saut de page (form feed)
\n	saut de ligne (newline)
\0ddd	caractère ASCII ddd (octal)
\xdd	caractère ASCII dd (hexadécimal)
\udddd	caractère Unicode dddd (hexadécimal)

3.10.2. L'addition de chaines

Java admet l'opérateur + comme opérateur de concaténation de chaines de caractères.

L'opérateur + permet de concatener plusieurs chaines. Il est possible d'utiliser l'opérateur +=

Exemple :

```
String texte = « »;
texte += « Hello »;
texte += « World3 »;
```

Cet opérateur sert aussi à concatener des chaînes avec tous les types de bases. La variable ou constante est alors convertie en chaîne et ajoutée à la précédente. La condition préalable est d'avoir au moins une chaîne dans l'expression sinon le sinon '+' est évalué comme opérateur mathématique.

Exemple :

```
System.out.println(« La valeur de Pi est : »+Math.PI);
int duree = 121;
System.out.println(« durée = » +duree);
```

3.10.3. La comparaison de deux chaines

Il faut utiliser la méthode equals()

Exemple :

```
String texte1 = « texte 1 »;
String texte2 = « texte 2 »;
if ( texte1.equals(texte2) )...
```

3.10.4. La détermination de la longueur d'une chaine

La méthode length() permet de déterminer la longueur d'une chaine.

Exemple :

```
String texte = « texte »;
int longueur = texte.length();
```

3.10.5. La modification de la casse d'une chaine

Les méthodes Java toUpperCase() et toLowerCase() permettent respectivement d'obtenir une chaîne tout en majuscule ou tout en minuscule.

Exemple :

```
String texte = « texte »;
String textemaj = texte.toUpperCase();
```

4. La programmation orientée objet

L'idée de base de la programmation orientée objet est de rassembler dans une même entité appelée objet les données et les traitements qui s'y appliquent.

Ce chapitre contient plusieurs sections :

- Le concept de classe : cette section présente le concept et la syntaxe de la déclaration d'une classe
- Les objets : cette section présente la création d'un objet, sa durée de vie, le clonage d'objets, les références et la comparaison d'objets, l'objet null, les variables de classes, la variable this et l'opérateur instanceof.
- Les modificateurs d'accès : cette section présente les modificateurs d'accès des entités classes, méthodes et attributs ainsi que les mots clés qui permettent de qualifier ces entités
- Les propriétés ou attributs : cette section présente les données d'une classe : les propriétés ou attributs
- Les méthodes : cette section présente la déclaration d'une méthode, la transmissions de paramètres, l'emmission de messages, la surcharge, la signature d'une méthode et le polymorphisme et des méthodes particulières : les constructeurs, le destructeur et les accesseurs
- L'héritage : cette section présente l'héritage : son principe, sa mise en oeuvre, ses conséquences. Il présente aussi la redéfinition d'une méthode héritée et les interfaces
- Les packages : cette section présente la définition et l'utilisation des packages
- Les classes internes : cette section présente une extension du langage Java qui permet de definir une classe dans une autre.
- La gestion dynamique des objets : cette section présente rapidement la gestion dynamique des objets grace à l'introspection

4.1. Le concept de classe

Une classe est le support de l 'encapsulation : c'est un ensemble de données et de fonctions regroupées dans une même entité. Une classe est une description abstraite d'un objet. Les fonctions qui opèrent sur les données sont appelées des méthodes. Instancier une classe consiste à créer un objet sur son modèle. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type et variable.

Java est un langage orienté objet : tout appartient à une classe sauf les variables de type primitives.

Pour accéder à une classe il faut en déclarer une instance de classe ou objet.

Une classe comporte sa déclaration, des variables et la définition de ses méthodes.

Une classe se compose en deux parties : un en−tête et un corps. Le corps peut être divisé en 2 sections : la déclaration des données et des constantes et la définition des méthodes. Les méthodes et les données sont pourvues d'attributs de visibilité qui gère leur accessibilité par les composants hors de la classe.

4.1.1. La syntaxe de déclaration d'une classe

modificateurs nom_de_classe [extends classe_mere] [implements interface] { ... }

```
ClassModifiers class ClassName [extends SuperClass] [implemenents Interfaces]
```

```
{
   // insérer ici les champs et les méthodes
}
```

Les modificateurs de classe (ClassModifiers) sont :

Modificateur	Role
abstract	la classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.
final	la classe ne peut pas être modifiée, sa redéfinition grace à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.
private	la classe n'est accessible qu'à partir du fichier où elle est définie
public	La classe est accessible partout

Les modificateurs abstract et final ainsi que public et private sont mutuellement exclusifs.

Le mot clé **extends** permet de spécifier une superclasse éventuelle : ce mot clé permet de préciser la classe mère dans une relation d'héritage.

Le mot clé **implements** permet de spécifier une ou des interfaces que la classe implémente. Cela permet de récupérer quelques avantages de l'héritage multiple.

L'ordre des méthodes dans une classe n'a pas d'importance. Si dans une classe, on rencontre d'abord la méthode A puis la méthode B, B peut être appelée sans problème dans A.

4.2. Les objets

Les objets contiennent des attributs et des méthodes. Les attributs sont des variables ou des objets nécessaires au fonctionnement de l'objet. En Java, une application est un objet. La classe est la description d'un objet. Un objet est une instance d'une classe. Pour chaque instance d'une classe, le code est le même, seules les données sont différentes à chaque objet.

4.2.1. La création d'un objet : instancier une classe

Il est nécessaire de définir la déclaration d'une variable ayant le type de l'objet désiré. La déclaration est de la forme nom_de_classe nom_de_variable

Exemple :

```
MaClasse m;
String chaine;
```

L'opérateur new se charge de créer une instance de la classe et de l'associer à la variable

Exemple :

```
m = new MaClasse();
```

Il est possible de tout reunir en une seule déclaration

Exemple :

```
MaClasse m = new MaClasse();
```

Chaque instance d'une classe nécessite sa propre variable. Plusieurs variables peuvent désigner un même objet.

En Java, tous les objets sont instanciés par allocation dynamique. Dans l'exemple, la variable m contient une référence sur l'objet instancié (contient l'adresse de l'objet qu'elle désigne : attention toutefois, il n'est pas possible de manipuler ou d'effectuer des opérations directement sur cette adresse comme en C).

Si m2 désigne un objet de type MaClasse, l'instruction m2 = m ne définit pas un nouvel objet mais m et m2 désignent tous les deux le même objet.

L'opérateur new est un opérateur de haute priorité qui permet d'instancier des objets et d'appeler une méthode particulière de cet objet : le contructeur. Il fait appel à la machine virtuelle pour obtenir l'espace mémoire nécessaire à la représentation de l'objet puis appelle le constructeur pour initialiser l'objet dans l'emplacement obtenu. Il renvoie une valeur qui référence l'objet instancié.

Si l'opérateur new n'obtient pas l'allocation mémoire nécessaire, il lève l'exception OutOfMemoryError.

Remarque sur les objets de type String : un objet String est automatiquement créé lors de l'utilisation d'une constante chaîne de caractères sauf si celle ci est déjà utilisée dans la classe. Ceci permet une simplification lors de la compilation de la classe.

Exemple :

```
public class TestChaines1 {

  public static void main(String[] args) {
    String chaine1 = "bonjour";
    String chaine2 = "bonjour";
    System.out.println("(chaine1 == chaine2) = " + (chaine1 == chaine2) );
  }
}
```

Résultat :

```
(chaine1 == chaine2) = true
```

Pour obtenir une seconde instance de la chaine, il faut explicitement demander sa création en utilisant l'opérateur new.

Exemple :

```
public class TestChaines2 {

  public static void main(String[] args) {
    String chaine1 = "bonjour";
    String chaine2 = new String("bonjour");
    System.out.println("(chaine1 == chaine2) = " + (chaine1 == chaine2) );
  }
}
```

Résultat :

```
(chaine1 == chaine2) = false
```

Remarque : les tests réalisés dans ces deux exemples sont réalisés sur les références des instances. Pour tester l'égalité de la valeur des chaînes, il faut utiliser la méthode equals() de la classe String.

4.2.2. La durée de vie d'un objet

Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.

La durée de vie d'un objet passe par trois étapes :

- la déclaration de l'objet et l'instanciation grace à l'opérateur new

Exemple :

```
nom_de_classe nom_d_objet = new nom_de_classe( ... );
```

- l'utilisation de l'objet en appelant ces méthodes
- la suppression de l'objet : elle est automatique en Java grace à la machine virtuelle. La restitution de la mémoire inutilisée est prise en charge par le récupérateur de mémoire (garbage collector). Il n'existe pas d'instruction delete comme en C++.

4.2.3. La création d'objets identiques

Exemple :

```
MaClasse m1 = new MaClasse();
MaClasse m2 = m1;
```

m1 et m2 contiennent la même référence et pointent donc tous les deux sur le même objet : les modifications faites à partir d'une des variables modifient l'objet.

Pour créer une copie d'un objet, il faut utiliser la méthode clone() : cette méthode permet de créer un deuxième objet indépendant mais identique à l'original. Cette méthode est héritée de la classe Object qui est la classe mère de toute les classes en Java.

Exemple :

```
MaClasse m1 = new MaClasse();
MaClasse m2 = m1.clone();
```

m1 et m2 ne contiennent plus la même référence et pointent donc sur des objets différents.

4.2.4. Les références et la comparaison d'objets

Les variables de type objet que l'on déclare ne contiennent pas un objet mais une référence vers cet objet. Lorque l'on écrit c1 = c2 (c1 et c2 sont des objets), on copie la référence de l'objet c2 dans c1 : c1 et c2 réfèrent au même objet (ils pointent sur le même objet). L'opérateur == compare ces références. Deux objets avec des propriétés identiques sont deux objets distincts :

Exemple :

```
Rectangle r1 = new Rectangle(100,50);
Rectangle r2 = new Rectangle(100,50);
if (r1 == r1) { ... } // vrai
if (r1 == r2) { ... } // faux
```

Pour comparer l'égalité des variables de deux instances, il faut munir la classe d'une méthode à cet effet : la méthode equals héritée de Object.

Pour s'assurer que deux objets sont de la même classe, il faut utiliser la méthode getClass() de la classe Object dont toutes les classes héritent.

Exemple :

```
(obj1.getClass().equals(obj2.getClass())
```

4.2.5. L'objet null

L'objet null est utilisable partout. Il n'appartient pas à une classe mais il peut être utilisé à la place d'un objet de n'importe quelle classe ou comme paramètre. null ne peut pas être utilisé comme un objet normal : il n'y a pas d'appel de méthodes et aucune classe ne peut en hériter.

Le fait d'initialiser une variable référant un objet à null permet au ramasse miette de libérer la mémoire allouée à l'objet.

4.2.6. Les variables de classes

Elles ne sont définies qu'une seule fois quel que soit le nombre d'objets instanciés de la classe. Leur déclaration est accompagnée du mot clé static

Exemple :

```
public class MaClasse() {
    static int compteur = 0;
}
```

L'appartenance des variables de classe à une classe entière et non à un objet spécifique permet de remplacer le nom de la variable par le nom de la classe.

Exemple :

```
MaClasse m = new MaClasse();
int c1 = m.compteur;
int c2 = MaClasse.compteur;
```

c1 et c2 possèdent la même valeur.

Ce type de variable est utile pour par exemple compter le nombre d'instanciation de la classe qui est faite.

4.2.7. La variable this

Cette variable sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation. this est un objet qui est égale à l'instance de l'objet dans lequel il est utilisé.

Exemple :

```
private int nombre;
public maclasse(int nombre) {
   nombre = nombre; // variable de classe = variable en paramètre du constructeur
}
```

Il est préférable d'écrire

```
this.nombre = nombre;
```

Cette référence est habituellement implicite :

Exemple :

```
class MaClasse() {
  String chaine = « test » ;
  Public String getChaine() { return chaine) ;
  // est équivalent à public String getChaine (this.chaine);
}
```

This est aussi utilisé quand l'objet doit appeler une méthode en se passant lui même en paramètre de l'appel.

4.2.8. L'opérateur instanceof

L'opérateur instanceof permet de déterminer la classe de l'objet qui lui est passé en paramètre. La syntaxe est objet instanceof classe

Exemple :

```
void testClasse(Object o) {
   if (o instanceof MaClasse )
      System.out.println(« o est une instance de la classe MaClasse »);
   else System.out.println(« o n'est pas un objet de la classe MaClasse »);
}
```

Il n'est toutefois pas possible d'appeler une méthode de l'objet car il est passé en paramètre avec un type Object

Exemple :

```
void afficheChaine(Object o) {
    if (o instanceof MaClasse)
        System.out.println(o.getChaine());
        // erreur à la compil car la méthode getChaine()
        //n'est pas définie dans la classe Object
}
```

Pour résoudre le problème, il faut utiliser la technique du casting (conversion).

Exemple :

```
void afficheChaine(Object o) {
   if (o instanceof MaClasse)
   {
      MaClasse m = (MaClasse) o;
      System.out.println(m.getChaine());
      // OU System.out.println( ((MaClasse) o).getChaine() );
   }
}
```

4.3. Les modificateurs d'accès

Ils se placent avant ou après le type de l'objet mais la convention veut qu'ils soient placés avant.

Ils s'appliquent aux classes et/ou aux méthodes et/ou aux attributs.

Ils ne peuvent pas être utilisés pour qualifier des variables locales : seules les variables d'instances et de classes peuvent en profiter.

Ils assurent le contrôle des conditions d'héritage, d'accès aux éléments et de modification de données par les autres objets.

4.3.1. Les mots clés qui gèrent la visibilité des entités

De nombreux langages orientés objet introduisent des attributs de visibilité pour reglémenter l'accès aux classes et aux objets, aux méthodes et aux données.

Il existe 3 modificateurs qui peuvent être utilisés pour définir les attributs de visibilité des entités (classes, méthodes ou attributs) : public, private et protected. Leur utilisation permet de définir des niveaux de protection différents (présentés dans un ordre croissant de niveau de protection offert) :

Modificateur	Role
public	Une variable, méthode ou classe déclarée public est visible par tout les autres objets. Dans la version 1.0, une seule classe public est permise par fichier et son nom doit correspondre à celui du fichier. Dans la philosophie orientée objet aucune donnée d'une classe ne devrait être déclarée publique : il est préférable d'écrire des méthodes pour la consulter et la modifier
par défaut : package friendly	Il n'existe pas de mot clé pour définir ce niveau, qui est le niveau par défaut lorsqu'aucun modificateur n'est précisé. Cette déclaration permet à une entité (classe, méthode ou variable) d'être visible par toutes les classes se trouvant dans le même package.
protected	Si une classe, une méthode ou une variable est déclarée protected , seules les méthodes présentes dans le même package que cette classe ou ses sous classes pourront y acceder. On ne peut pas qualifier une classe avec protected.
private	C'est le niveau de protection le plus fort. Les composants ne sont visibles qu'à l'intérieur de la classe : ils ne peuvent être modifiés que par des méthodes définies dans la classe prévues à cet effet. Les méthodes déclarées private ne peuvent pas être en même temps déclarée abstract car elles ne peuvent pas être redéfinies dans les classes filles.

Ces modificateurs d'accès sont mutuellement exclusifs.

4.3.2. Le mot clé static

Le mot clé static s'applique aux variables et aux méthodes.

Les variables d'instance sont des variables propres à un objet. Il est possible de définir une variable de classe qui est partagée entre toutes les instances d'une même classe : elle n'existe donc qu'une seule fois en mémoire. Une telle variable permet de stocker une constante ou une valeur modifiée tour à tour par les instances de la classe. Elle se définit avec le mot clé static.

Exemple :

```
public class Cercle {
    static float pi = 3.1416f;
    float rayon;
    public Cercle(float rayon) { this.rayon = rayon; }
    public float surface() { return rayon * rayon * pi;}
}
```

Il est aussi possible par exemple de mémoriser les valeurs min et max d'un ensemble d'objets de même classe.

Une méthode static est une méthode qui n'agit pas sur des variables d'instance mais uniquement sur des variables de classe. Ces méthodes peuvent être utilisées sans instancier un objet de la classe. Les méthodes ainsi définies peuvent être appelées avec la notation classe.methode() au lieu de objet.methode() : la première forme est fortement recommandée pour éviter toute confusion.

Il n'est pas possible d'appeler une méthode d'instance ou d'accéder à une variable d'instance à partir d'une méthode de classe statique.

4.3.3. Le mot clé final

Le mot clé final s'applique aux variables de classe ou d'instance, aux méthodes et aux classes. Il permet de rendre l'entité sur laquelle il s'applique non modifiable une fois qu'elle est déclarée pour une méthode ou une classe et initialisée pour une variable.

Une variable qualifiée de final signifie que la valeur de la variable ne peut plus être modifée une fois que celle ci est initialisée. On ne peut pas déclarer de variables final locales à une méthode.

Exemple :

```
package com.moi.test;

public class Constante2 {

  public final int constante;

  public Constante2() {
    this.constante = 10;
  }

}
```

Une fois la variable déclarée final initialisée, il n'est plus possible de modifier sa valeur. Une vérification est opérée par le compilateur.

Exemple :

```
package com.moi.test;

public class Constante1 {

  public static final int constante = 0;

  public Constante1() {
    this.constante = 10;
  }

}
```

Exemple :

```
C:\>javac Constante1.java
Constante1.java:6: cannot assign a value to final variable constante
    this.constante = 10;
    ^
1 error
```

Les constantes sont qualifiées avec les modificateurs final et static.

Exemple :

```
public static final float PI = 3.141f;
```

Une méthode déclarée final ne peut pas être redéfinie dans une sous classe. Une méthode possédant le modificateur final pourra être optimisée par le compilateur car il est garanti qu'elle ne sera pas sous classée.

Lorsque le modificateur final est ajouté à une classe, il est interdit de créer une classe qui en hérite.

Pour une méthode ou une classe, on renonce à l'héritage mais ceci peut s'avérer nécessaire pour des questions de sécurité ou de performance. Le test de validité de l'appel d'une méthode est bien souvent repoussé à l'exécution, en fonction du type de l'objet appelé (c'est la notion de polymorphisme qui sera détaillée ultérieurement). Ces tests ont un coût en terme de performance.

4.3.4. Le mot clé abstract

Le mot cle abstract s'applique aux méthodes et aux classes.

Abstract indique que la classe ne pourra être instanciée telle quelle. De plus, toutes les méthodes de cette classe abstract ne sont pas implémentées et devront être redéfinies par des méthodes complètes dans ses sous classes.

Abstract permet de créer une classe qui sera une sorte de moule. Toutes les classes dérivées pouront profiter des méthodes héritées et n'auront à implémenter que les méthodes déclarées abstract.

Exemple :

```
abstract class ClasseAbstraite {
    ClasseBastraite() { ... //code du constructeur }
    void méthode() { ... // code partagé par tous les descendants}
    abstract void méthodeAbstraite();
}

class ClasseComplete extends ClasseAbstraite {
    ClasseComplete() { super(); ... }
    void méthodeAbstraite() { ... // code de la méthode }
    // void méthode est héritée
}
```

Une méthode abstraite est une méthode déclarée avec le modificateur abstract et sans corps. Elle correspond à une méthode dont on veut forcer l'implémentation dans une sous classe. L'abstraction permet une validation du codage : une sous classe sans le modificateur abstract et sans définition explicite d'une ou des méthodes abstraites génère une erreur de compilation.

Une classe est automatiquement abstraite dès qu'une de ses méthodes est déclarée abstraite. Il est possible de définir une classe abstraite sans méthodes abstraites.

4.3.5. Le mot clé synchronized

Permet de gérer l'accès concurrent aux variables et méthodes lors de traitement de thread (exécution « simultanée » de plusieurs petites parties de code du programme)

4.3.6. Le mot clé volatile

Le mot cle volatile s'applique aux variables.

Précise que la variable peut être changée par un périphérique ou de manière asynchrone. Cela indique au compilateur de ne pas stocker cette variable dans des registres. A chaque utilisation, on lit la valeur et on réécrit immédiatement le résultat s'il a changé.

4.3.7. Le mot clé native

Une méthode native est une méthode qui est implémentée dans un autre langage. L'utilisation de ce type de méthode limite la portabilité du code mais permet une vitesse exécution plus rapide.

4.4. Les propriétés ou attributs

Les données d'une classe sont contenues dans des variables nommées propriétés ou attributs. Ce sont des variables qui peuvent être des variables d'instances, des variables de classes ou des constantes.

4.4.1. Les variables d'instances

Une variable d'instance nécessite simplement une déclaration de la variable dans le corps de la classe.

Exemple :

```
public class MaClasse {
   public int valeur1 ;
   int valeur2 ;
   protected int valeur3 ;
   private int valeur4 ;
}
```

Chaque instance de la classe a accès à sa propre occurrence de la variable.

4.4.2. Les variables de classes

Les variables de classes sont définies avec le mot clé static

Exemple (code Java 1.1) :

```
public class MaClasse {
   static int compteur ;
}
```

Chaque instance de la classe partage la même variable.

4.4.3. Les constantes

Les constantes sont définies avec le mot clé final : leur valeur ne peut pas être modifiée une fois qu'elles sont initialisées..

Exemple (code Java 1.1) :

```
public class MaClasse {
   final double pi=3.14 ;
}
```

4.5. Les méthodes

Les méthodes sont des fonctions qui implémentent les traitements de la classe.

4.5.1. La syntaxe de la déclaration

La syntaxe de la déclaration d'une méthode est :

```
modificateurs type_retourné nom_méthode ( arg1, ... ) {... }
    // définition des variables locales et du bloc d'instructions
}
```

Le type retourné peut être élémentaire ou correspondre à un objet. Si la méthode ne retourne rien, alors on utilise **void**.

Le type et le nombre d'arguments déclarés doivent correspondre au type et au nombre d'arguments transmis. Il n'est pas possible d'indiquer des valeurs par défaut dans les paramètres. Les arguments sont passés par valeur : la méthode fait une copie de la variable qui lui est locale. Lorsqu'un objet est transmis comme argument à une méthode, cette dernière reçoit une référence qui désigne son emplacement mémoire d'origine et qui est une copie de la variable . Il est possible de modifier l'objet grace à ces méthodes mais il n'est pas possible de remplacer la référence contenue dans la variable passée en paramètre : ce changement n'aura lieu que localement à la méthode.

Les modificateurs de méthodes sont :

Modificateur	Role
public	la méthode est accéssible aux méthodes des autres classes
private	l'usage de la méthode est réservé aux autres méthodes de la même classe
protected	la méthode ne peut être invoquée que par des méthodes de la classe ou de ses sous classes
final	la méthode ne peut être modifiée (redifinition lors de l'héritage interdite)
static	la méthode appartient simultanément à tous les objets de la classe (comme une constante déclarée à l'intérieur de la classe). Il est inutile d'instancier la classe pour appeler la méthode mais la méthode ne peut pas manipuler de variable d'instance. Elle ne peut utiliser que des variables de classes.
synchronized	la méthode fait partie d'un thread. Lorsqu'elle est appelée, elle barre l'accès à son instance. L'instance est à nouveau libérée à la fin de son exécution.
native	le code source de la méthode est écrit dans un autre langage

Sans modificateur, la méthode peut être appelée par toutes autres méthodes des classes du package auquel appartient la classe.

La valeur de retour de la méthode doit être transmise par l'instruction return. Elle indique la valeur que prend la méthode et termine celle ci : toutes les instructions qui suivent return sont donc ignorées.

Exemple :

```
int add(int a, int b) {
   return a + b;
}
```

Il est possible d'inclure une instruction return dans une méthode de type void : cela permet de quitter la méthode.

La méthode main() de la classe principale d'une application doit être déclarée de la façon suivante :

```
public static void main (String args[]) { ... }
```

Exemple :

```
public class MonApp1 {
```

```
  public static void main(String[] args) {
    System.out.println("Bonjour");
  }
}
```

Cette déclaration de la méthode main() est imposée par la machine virtuelle pour reconnaitre le point d'entrée d'une application. Si la déclaration de la méthode main() diffère, une exception sera levée lors de la tentative d'exécution par la machine virtuelle.

Exemple :

```
public class MonApp2 {

  public static int main(String[] args) {
    System.out.println("Bonjour");
    return 0;
  }
}
```

Résultat :

```
C:\>javac MonApp2.java

C:\>java MonApp2
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Si la méthode retourne un tableau alors les [] peuvent être préciser après le type de retour ou après la liste des paramètres :

Exemple :

```
int[] valeurs() { ... }
int valeurs()[] { ... }
```

4.5.2. La transmission de paramètres

Lorsqu'un objet est passé en paramètre, ce n'est pas l'objet lui même qui est passé mais une référence sur l'objet. La référence est bien transmise par valeur et ne peut pas être modifiée mais l'objet peut être modifié via un message (appel d'une méthode).

Pour transmettre des arguments par référence à une méthode, il faut les encapsuler dans un objet qui prévoit les méthodes nécessaires pour les mises à jour.

Si un objet o transmet sa variable d'instance v en paramètre à une méthode m, deux situations sont possibles :

- si v est une variable primitive alors elle est passée par valeur : il est impossible de la modifier dans m pour que v en retour contienne cette nouvelle valeur.
- si v est un objet alors m pourra modifier l'objet en utilisant une méthode de l'objet passé en paramètre.

4.5.3. L'emmission de messages

Un message est émis lorsqu'on demande à un objet d'exécuter l'une de ses méthodes.

La syntaxe d'appel d'une méthode est : nom_objet.nom_méthode(parametre, ...) ;

Si la méthode appelée ne contient aucun paramètre, il faut laisser les parenthèses vides.

4.5.4. L'enchainement de références à des variables et à des méthodes

Exemple :

```
System.out.Println("bonjour");
```

Deux classes sont impliquées dans l'instruction : System et PrintStream. La classe System possède une variable nommée out qui est un objet de type PrintStream. Println() est une méthode de la classe PrintStream. L'instruction signifie : « utilise la méthode Println() de la variable out de la classe System ».

4.5.5. La surcharge de méthodes

La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des arguments différents. Le compilateur choisi la méthode qui doit être appellée en fonction du nombre et du type des arguments. Ceci permet de simplifier l'interface des classes vis à vis des autres classes.

Une méthode est surchargée lorqu'elle exécute des actions différentes selon le type et le nombre de paramètres transmis.

Il est donc possible de donner le même nom à deux méthodes différentes à condition que les signatures de ces deux méthodes soient différentes. La signature d'une méthode comprend le nom de la classe, le nom de la méthode et les types des paramètres.

Exemple :

```
class affiche{
    public void afficheValeur(int i) {
        System.out.println(« nombre entier = » + i);
    }

    public void afficheValeur(float f) {
        System.out.println(« nombre flottant = » + f);
    }
}
```

Il n'est pas possible d'avoir deux méthodes de même nom dont tous les paramètres sont identiques et dont seul le type retourné diffère.

Exemple :

```
class Affiche{

    public float convert(int i){
         return((float) i);
    }

    public double convert(int i){
        return((double) i);
    }
}
```

Résultat à la compilation :

```
C:\>javac Affiche.java
Affiche.java:5: Methods can't be redefined with a different return type: double
convert(int) was float convert(int)
public double convert(int i){
```

```
       ^
1 error
```

4.5.6. Les constructeurs

La déclaration d'un objet est suivie d'une sorte d'initialisation par le moyen d'une méthode particulière appelée constructeur pour que les variables aient une valeur de départ. Elle n'est systématiquement invoquée que lors de la création d'un objet.

Le constucteur suit la définition des autres méthodes excepté que son nom doit obligatoirement correspondre à celui de la classe et qu'il n'est pas typé, pas même void, donc il ne peut pas y avoir d'instruction return dans un constructeur. On peut surcharger un constructeur.

La définition d'un constructeur est facultative. Si elle n'est pas définie, la machine virtuelle appelle un constructeur par défaut vide créé automatiquement. Dès qu'un constructeur est explicitement défini, Java considère que le programmeur prend en charge la création des constructeurs et que le mécanisme par défaut, qui correspond à un constructeur sans paramètres, est supprimé. Si on souhaite maintenir ce mécanisme, il faut définir explicitement un constructeur sans paramètres.

Il existe plusieurs manières de définir un constructeur :

1. le constructeur simple : ce type de constructeur ne nécessite pas de définition explicite : son existence découle automatiquement de la définition de la classe.

Exemple :

```
public MaClasse() {}
```

2. le constructeur avec initialisation fixe : il permet de créer un constructeur par défaut

Exemple :

```
public MaClasse() {
    nombre = 5;
}
```

3. le constructeur avec initialisation des variables : pour spécifier les valeurs de données à initialiser on peut les passer en paramètres au constructeur

Exemple :

```
public MaClasse(int valeur) {
     nombre = valeur;
}
```

4.5.7. Le destructeur

Un destructeur permet d'exécuter du code lors de la libération, par le garbage collector, de l'espace mémoire occupé par l'objet. En Java, les destructeurs appellés finaliseurs (finalizers), sont automatiquement appellés par le garbage collector.

Pour créer un finaliseur, il faut redéfinir la méthode finalize() héritée de la classe Object.

Attention : selon l'implémentation du garbage collector dans la machine virtuelle, il n'est pas possible de prévoir le moment ou un objet sera traité par le garbage collector. De plus, l'appel du finaliseur n'est pas garanti : par exemple, si la machine virtuelle est brusquement arrêtée par l'utilisateur, le ramasse miette ne liberera pas la mémoire des objets en cours d'utilisation et les finaliseurs de ces objets ne seront pas appelés.

4.5.8. Les accesseurs

L'encapsulation permet de sécuriser l'accès aux données d'une classe. Ainsi, les données déclarées private à l'intérieur d'une classe ne peuvent être accédées et modifiées que par des méthodes définies dans la même classe. Si une autre classe veut accéder aux données de la classe, l'opération n'est possible que par l'intermédiaire d'une méthode de la classe prévue à cet effet. Ces appels de méthodes sont appelés « échanges de message ».

Un accesseur est une méthode publique qui donne l'accès à une variable d'instance privée. Pour une variable d'instance, il peut ne pas y avoir d'accesseur, un seul accesseur en lecture ou un accesseur en lecture et un autre en écriture. Par convention, les accesseurs en lecture commencent par get et les accesseurs en écriture commencent par set.

Exemple :

```
private int valeur = 13;

public int getValeur(){
    return(valeur);
}

public void setValeur(int val) {
    valeur = val;
}
```

Pour un attribut de type booléen, il est possible de faire commencer l'accesseur en lecture par is au lieu de get.

4.6. L'héritage

L'héritage est un mécanisme qui facilite la réutilisation du code et la gestion de son évolution. Elle définit une relation entre deux classes :

- une classe mère ou super classe
- une classe fille ou sous classe qui hérite de sa classe mère

4.6.1. Le principe de l'héritage

Grace à l'heritage, les objets d'une classe fille ont accès aux données et aux méthodes de la classe parent et peuvent les étendre. Les sous classes peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent. Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge.

L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de super classes et de sous classes. Une classe qui hérite d'une autre est une sous classe et celle dont elle hérite est une super classe. Une classe peut avoir plusieurs sous classes. Une classe ne peut avoir qu'une seule classe mère : il n'y a pas d'héritage multiple en Java.

Object est la classe parente de toutes les classes en Java. Toutes les variables et méthodes contenues dans Object sont accessibles à partir de n'importe quelle classe car par héritage successif toutes les classes héritent d'Object.

4.6.2. La mise en oeuvre de l'héritage

On utilise le mot clé extends pour indiquer qu'une classe hérite d'une autre. En l'absence de ce mot réservé associé à une classe, le compilateur considère la classe Object comme classe parent.

Exemple :

```
class Fille extends Mere { ... }
```

Pour invoquer une méthode d'une classe parent, il suffit d'indiquer la méthode préfixée par super. Pour appeler le constructeur de la classe parent il suffit d'écrire super(paramètres) avec les paramètres adéquats.

Le lien entre une classe fille et une classe parent est géré par le langage : une évolution des règles de gestion de la classe parent conduit à modifier automatiquement la classe fille dès que cette dernière est recompilée.

En Java, il est obligatoire dans un constructeur d'une classe fille de faire appel explicitement ou implicitement au constructeur de la classe mère.

4.6.3. L'accès aux propriétés héritées

Les variables et méthodes définies avec le modificateur d'accès public restent publiques à travers l'héritage et toutes les autres classes.

Une variable d'instance définie avec le modificateur private est bien héritée mais elle n'est pas accessible directement mais via les méthodes héritées.

Si l'on veut conserver pour une variable d'instance une protection semblable à celle assurée par le modificateur private, il faut utiliser le modificateur protected. La variable ainsi définie sera hérité dans toutes les classes descendantes qui pourront y accéder librement mais ne sera pas accessible hors de ces classes directement.

4.6.4. La redéfinition d'une méthode héritée

La redéfinition d'une méthode héritée doit impérativement conserver la déclaration de la méthode parent (type et nombre de paramètres, la valeur de retour et les exceptions propagées doivent être identique).

Si la signature de la méthode change, ce n'est plus une redéfinition mais une surcharge. Cette nouvelle méthode n'est pas héritée : la classe mere ne possède pas de méthode possédant cette signature.

4.6.5. Le polymorphisme

Le polymorphisme est la capacité, pour un même message de correspondre à plusieurs formes de traitements selon l'objet auquel ce message est adressé. La gestion du polymorphisme est assurée par la machine virtuelle dynamiquement à l'exécution.

4.6.6. Le transtypage induit par l'héritage facilitent le polymorphisme

L'héritage définit un cast implicite de la classe fille vers la classe mere : on peut affecter à une référence d'une classe n'importe quel objet d'une de ses sous classes.

Exemple : la classe Employe hérite de la classe Personne

```
   Personne p = new Personne («Dupond», «Jean»);
```

```
    Employe e = new Employe(«Durand», «Julien», 10000);
    p = e ; // ok : Employe est une sous classe de Personne
    Objet obj;
    obj = e ; // ok : Employe herite de Personne qui elle même hérite de Object
```

Il est possible d'écrire le code suivant si Employe hérite de Personne

Exemple :

```
   Personne[] tab = new Personne[10];
    tab[0]:= new Personne( «Dupond»,«Jean»);
    tab[1]:= new Employe(«Durand», «Julien», 10000);
```

Il est possible de surcharger une méthode héritée : la forme de la méthode à exécuter est choisie en fonction des paramètres associés à l'appel.

Compte tenu du principe de l'héritage, le temps d'exécution du programme et la taille du code source et de l'exécutable augmentent.

4.6.7. Les interfaces et l'héritage multiple

Avec l'héritage multiple, une classe peut hériter en même temps de plusieurs super classes. Ce mécanisme n'existe pas en Java. Les interfaces permettent de mettre en oeuvre un mécanisme de remplacement.

Une interface est un ensemble de constantes et de déclarations de méthodes correspondant un peu à une classe abstraite. C'est une sorte de standard auquel une classe peut répondre. Tous les objets qui se conforment à cette interface (qui implémentent cette interface) possèdent les méthodes et les constantes déclarée dans celle−ci. Plusieurs interfaces peuvent être implémentées dans une même classe.

Les interfaces se déclarent avec le mot cle interface et sont intégrées aux autres classes avec le mot clé implements. Une interface est implicitement déclarée avec le modificateur abstract.

Déclaration d'une interface :

```
[public] interface nomInterface [extends nomInterface1, nomInterface2 ... ] {
    // insérer ici des méthodes ou des champs static
}
```

Implémentation d'une interface :

```
Modificateurs class nomClasse [extends superClasse]
   [implements nomInterface1, nomInterface 2, ...] {
    //insérer ici des méthodes et des champs

}
```

Exemple :

```
interface AfficheType {
    void afficherType();
}

class Personne implements AfficheType {

    public void afficherType() {
        System.out.println(« Je suis une personne »);
    }
}
```

```
class Voiture implements AfficheType {

    public void afficherType() {

        System.out.println(« Je suis une voiture »);
    }
}
```

Exemple : déclaration d'un interface à laquelle doit se conformer tout individus

```
interface Individu {
    String getNom();
    String getPrenom();
    Date getDateNaiss();
}
```

Toutes les méthodes d'une interface sont abstraites : elles sont implicitement déclarées comme telles.

Une interface peut être d'accès public ou package. Si elle est publique, toutes ses méthodes sont implicitement publiques même si elles ne sont pas déclarées avec le modificateur public. Si elle est d'accès package, il s'agit d'une interface d'implémentation pour les autres classes du package et ses méthodes ont le même accès package : elles sont accessibles à toutes les classes du packages.

Les seules variables que l'on peut définir dans une interface sont des variables de classe qui doivent être constantes : elles sont donc implicitement déclarées avec le modificateur static et final même si elles sont définies avec d'autres modificateurs.

Exemple

```
public interface MonInterface {
  public int VALEUR=0;
  void maMethode();
}
```

Toute classe qui implémente cette interface doit au moins posséder les méthodes qui sont déclarées dans l'interface. L'interface ne fait que donner une liste de méthodes qui seront à définir dans les classes qui implémentent l'interface.

Les méthodes déclarées dans une interface publique sont implicitement publiques et elles sont héritées par toutes les classes qui implémentent cette interface. Une telle classe doit, pour être instanciable, définir toutes les méthodes héritées de l'interface.

Une classe peut implémenter une ou plusieurs interfaces tout en héritant de sa classe mère.

L'implémentation d'une interface définit un cast : l'implémentation d'une interface est une forme d'héritage. Comme pour l'héritage d'une classe, l'héritage d'une classe qui implémente une interface définit un cast implicite de la classe fille vers cette interface. Il est important de noter que dans ce cas il n'est possible de faire des appels qu'à des méthodes de l'interface. Pour utiliser des méthodes de l'objet, il faut définir un cast explicite : il est préférable de contrôler la classe de l'objet pour éviter une exception ClassCastException à l'éxecution

4.6.8. Des conseils sur l'héritage

Lors de la création d'une classe « mère » il faut tenir compte des points suivants :

- la définition des accès aux variables d'instances, très souvent privées, doit être réfléchie entre protected et private
- pour empecher la redéfinition d'une méthode (surcharge) il faut la déclarer avec le modificateur final

Lors de la création d'une classe fille, pour chaque méthode héritée qui n'est pas final, il faut envisager les cas suivants :

- la méthode héritée convient à la classe fille : on ne doit pas la redéfinir
- la méthode héritée convient mais partiellement du fait de la spécialisation apportée par la classe fille : il faut la redéfinir voir la surcharger. La plupart du temps une redéfinition commencera par appeler la méthode héritée (via super) pour garantir l 'évolution du code
- la méthode héritée ne convient pas : il faut redéfinir ou surcharger la méthode sans appeler la méthode héritée lors de la redéfinition.

4.7. Les packages

4.7.1. La définition d'un package

En Java, il existe un moyen de regrouper des classes voisines ou qui couvrent un même domaine : ce sont les packages. Pour réaliser un package, on écrit un nombre quelconque de classes dans plusieurs fichiers d'un même repertoire et au début de chaque fichier on met la directive ci dessous ou nom−du−package doit être identique au nom du répertoire :

package nomPackage;

La hiérarchie d'un package se retrouve dans l'arborescence du disque dur puisque chaque package est dans un répertoire nommé du nom du package.

Remarque : Il est préférable de laisser les fichiers source .java avec les fichiers compilés .class

D'une façon générale, l'instruction package associe toutes les classes qui sont définies dans un fichier source à un même package.

Le mot clé package doit être la première instruction dans un fichier source et il ne doit être présent qu'une seule fois dans le fichier source (une classe ne peut pas appartenir à plusieurs packages).

4.7.2. L'utilisation d'un package

Pour utiliser ensuite le package ainsi créé, on l'importe dans le fichier :

import nomPackage.*;

Pour importer un package, il y a deux méthodes si le chemin de recherche est correctement renseigné : préciser un nom de classe ou interface qui sera l'unique entité importé ou mettre une * indiquant toutes les classes et interfaces définies dans le package.

Exemple	Role
`import nomPackage.*;`	toutes les classes du package sont importées
`import nomPackage.nomClasse;`	appel à une seule classe : l'avantage de cette notation est de réduire le temps de compilation

Attention : l'astérisque n'importe pas les sous paquetages. Par exemple, il n'est pas possible d'écrire import java.*.

Il est possible d'appeler une méthode d'un package sans inclure ce dernier dans l'application en precisant son nom complet :

nomPackage.nomClasse.nomméthode(arg1, arg2 ...)

Il existe plusieurs types de packages : le package par défaut (identifié par le point qui représente le répertoire courant et permet de localiser les classes qui ne sont pas associées à un package particulier), les packages standards qui sont empaquetés dans le fichier classes.zip et les packages personnels

Le compilateur implémente automatiquement une commande import lors de la compilation d'un programme Java même si elle ne figure pas explicitement au début du programme : import java.lang.*; Ce package contient entre autre les classes de base de tous les objets Java dont la classe Object.

Un package par défaut est systématiquement attribué par le compilateur aux classes qui sont définies sans déclarer explicitement une appartenance à un package. Ce package par défaut correspond au répertoire courant qui est le répertoire de travail.

4.7.3. La collision de classes.

Deux classes entre en collision lorsqu'elles portent le même nom mais qu'elles sont définies dans des packages différents. Dans ce cas, il faut qualifier explicitement le nom de la classe avec le nom complet du package.

4.7.4. Les packages et l'environnement système

Les classes Java sont importées par le compilateur (au moment de la compilation) et par la machine virtuelle (au moment de l'exécution). Les techniques de chargement des classes varient en fonction de l'implémentation de la machine virtuelle. Dans la plupart des cas, une variable d'environnement CLASSPATH référence tous les répertoires qui hébergent des packages succeptibles d'être importés.

Exemple sous Windows :

CLASSPATH = .;C:\Java\JDK\Lib\classes.zip; C:\rea_java\package

L'importation des packages ne fonctionne que si le chemin de recherche spécifié dans une variable particulière pointe sur les packages, sinon le nom du package devra refléter la structure du répertoire ou il se trouve. Pour déterminer l'endroit où se trouvent les fichiers .class à importer, le compilateur utilise une variable d'environnement dénommée CLASSPATH. Le compilateur peut lire les fichiers .class comme des fichiers indépendants ou comme des fichiers ZIP dans lesquels les classes sont reunies et compressées.

4.8. Les classes internes

Les classes internes (inner classes) sont une extension du langage Java introduite dans la version 1.1 du JDK. Ce sont des classes qui sont définies dans une autre classe. Les difficultés dans leur utilisation concerne leur visibilité et leur accès aux membres de la classe dans laquelle elles sont définies.

Exemple très simple :

```
public class ClassePrincipale1 {
  class ClasseInterne {
  }
}
```

Les classes internes sont particulièrement utiles pour :

- permettre de définir une classe à l'endroit ou une seule autre en a besoin
- définir des classes de type adapter (essentiellement à partir du JDK 1.1 pour traiter des évenements émis par les interfaces graphiques)

- définir des méthodes de type callback d'une façon générale

Pour permettre de garder une compatibilité avec la version précédente de la JVM, seul le compilateur a été modifié. Le compilateur interprète la syntaxe des classes internes pour modifier le code source et générer du byte code compatible avec la première JVM.

Il est possible d'imbriquer plusieurs classes internes. Java ne possèdent pas de restrictions sur le nombre de classes qu'il est ainsi possible d'imbriquer. En revanche une limitation peut intervenir au niveau du système d'exploitation en ce qui concerne la longueur du nom du fichier .class généré pour les différentes classes internes.

Si plusieurs classes internes sont imbriquées, il n'est pas possible d'utiliser un nom pour la classe qui soit déjà attribuée à une de ces classes englobantes. Le compilateur génèrera une erreur à la compilation.

Exemple :

```
public class ClassePrincipale6 {
  class ClasseInterne1 {
    class ClasseInterne2 {
      class ClasseInterne3 {
      }
    }
  }
}
```

Le nom de la classe interne utilise la notation qualifiée avec le point préfixé par le nom de la classe principale. Ainsi, pour utiliser ou accéder à une classe interne dans le code, il faut la préfixer par le nom de la classe principale suivi d'un point.

Cependant cette notation ne représente pas physiquement le nom du fichier qui contient le byte code. Le nom du fichier qui contient le byte code de la classe interne est modifié par le compilateur pour éviter des conflits avec d'autres noms d'entité : à partir de la classe principale, les points de séparation entre chaque classe interne sont remplacé par un caractère $ (dollar).

Par exemple, la compilation du code de l'exemple précédent génère quatres fichiers contenant le byte code :
ClassePrincipale6$ClasseInterne1$ClasseInterne2ClasseInterne3.class
ClassePrincipale6$ClasseInterne1$ClasseInterne2.class
ClassePrincipale6$ClasseInterne1.class
ClassePrincipale6.class

L'utilisation du signe $ entre la classe principale et la classe interne permet d'éviter des confusions de nom entre le nom d'une classe appartenant à un package et le nom d'une classe interne.

L'avantage de cette notation est de créer un nouvel espace de nommage qui dépend de la classe et pas d'un package. Ceci renforce le lien entre la classe interne et sa classe englobante.

C'est le nom du fichier qu'il faut préciser lorsque l'on tente de charger la classe avec la méthode forName() de la classe Class. C'est aussi sous cette forme qu'est restitué le résultat d'un appel aux méthodes getClass().getName() sur un objet qui est une classe interne.

Exemple :

```
public class ClassePrincipale8 {
  public class ClasseInterne {
  }

  public static void main(String[] args) {
    ClassePrincipale8 cp = new ClassePrincipale8();
    ClassePrincipale8.ClasseInterne ci = cp. new ClasseInterne() ;
    System.out.println(ci.getClass().getName());
  }
}
```

Resultat :

```
java ClassePrincipale8
ClassePrincipale8$ClasseInterne
```

L'accessibilité à la classe interne respecte les règles de visibilité du langage. Il est même possible de définir une classe interne private pour limiter son accès à sa seule classe principale.

Exemple :

```
public class ClassePrincipale7 {
        private class ClasseInterne {
        }
}
```

Il n'est pas possible de déclarer des membres statiques dans une classe interne :

Exemple :

```
public class ClassePrincipale10 {
        public class ClasseInterne {
                static int var = 3;
        }
}
```

Resultat :

```
javac ClassePrincipale10.java
ClassePrincipale10.java:3: Variable var can't be static in inner class ClassePri
ncipale10. ClasseInterne.  Only members of interfaces and top-level classes can
be static.
                static int var = 3;
                           ^
1 error
```

Pour pouvoir utiliser une variable de classe dans une classe interne, il faut la déclarer dans sa classe englobante.

Il existe quatres types de classes internes :

- les classes internes non statiques : elles sont membres à part entière de la classe qui les englobe et peuvent accéder à tous les membres de cette dernière
- les classes internes locales : elles sont définies dans un block de code. Elles peuvent être static ou non.
- les classes internes anonymes : elles sont définies et instanciées à la volée sans posséder de nom
- les classe internes statiques : elles sont membres à part entière de la classe qui les englogent et peuvent accéder uniquement aux membres statiques de cette dernière

4.8.1. Les classes internes non statiques

Les classes internes non statiques (member inner-classes) sont définies dans une classe dite " principale " (top-level class) en tant que membre de cette classe. Leur avantage est de pouvoir accéder aux autres membres de la classe principale même ceux déclarés avec le modificateur private.

Exemple :

```
public class ClassePrincipale20 {
        private int valeur = 1;

        class ClasseInterne {
                public void afficherValeur() {
                        System.out.println("valeur = "+valeur);
                }
        }
```

```
        public static void main(String[] args) {
                ClassePrincipale20 cp = new ClassePrincipale20();
                ClasseInterne ci = cp. new ClasseInterne();
                ci.afficherValeur();
        }
}
```

Resultat :

```
C:\testinterne>javac ClassePrincipale20.java

C:\testinterne>java ClassePrincipale20
valeur = 1
```

Le mot clé this fait toujours référence à l'instance en cours. Ainsi this.var fait référence à la variable var de l'instance courante. L'utilisation du mot clé this dans une classe interne fait donc référence à l'instance courante de cette classe interne.

Exemple :

```
public class ClassePrincipale16 {
        class ClasseInterne {
                int var = 3;

                public void affiche() {
                        System.out.println("var      = "+var);
                        System.out.println("this.var = "+this.var);
                }
        }

        ClasseInterne ci = this. new ClasseInterne();

        public static void main(String[] args) {
                ClassePrincipale16 cp = new ClassePrincipale16();
                ClasseInterne ci = cp. new ClasseInterne();
                ci.affiche();
        }
}
```

Resultat :

```
C:\>java ClassePrincipale16
var      = 3
this.var = 3
```

Une classe interne a accès à tous les membres de sa classe principale. Dans le code, pour pouvoir faire référence à un membre de la classe principale, il suffit simplement d'utiliser son nom de variable.

Exemple :

```
public class ClassePrincipale17 {
        int valeur = 5;

        class ClasseInterne {
                int var = 3;

                public void affiche() {
                        System.out.println("var      = "+var);
                        System.out.println("this.var = "+this.var);
                        System.out.println("valeur   = "+valeur);
                }
        }

        ClasseInterne ci = this. new ClasseInterne();

        public static void main(String[] args) {
```

```
                ClassePrincipale17 cp = new ClassePrincipale17();
                ClasseInterne ci = cp. new ClasseInterne();
                ci.affiche();
        }
}
```

Resultat :

```
C:\testinterne>java ClassePrincipale17
var      = 3
this.var = 3
valeur   = 5
```

La situation se complique un peu plus, si la classe principale et la classe interne possède tous les deux un membre de même nom. Dans ce cas, il faut utiliser la version qualifiée du mot clé this pour accéder au membre de la classe principale. La qualification se fait avec le nom de la classe principale ou plus généralement avec le nom qualifié d'une des classes englobantes.

Exemple :

```
public class ClassePrincipale18 {
        int var = 5;

        class ClasseInterne {
                int var = 3;

                public void affiche() {
                        System.out.println("var                          = "+var);
                        System.out.println("this.var                     = "+this.var);
                        System.out.println("ClassePrincipale18.this.var  = "
                          +ClassePrincipale18.this.var);
                }
        }

        ClasseInterne ci = this. new ClasseInterne();

        public static void main(String[] args) {
                ClassePrincipale18 cp = new ClassePrincipale18();
                ClasseInterne ci = cp. new ClasseInterne();
                ci.affiche();
        }
}
```

Resultat :

```
C:\>java ClassePrincipale18
var                          = 3
this.var                     = 3
ClassePrincipale18.this.var  = 5
```

Comme une classe interne ne peut être nommée du même nom que l'une de ces classes englobantes, ce nom qualifié est unique et il ne risque pas d'y avoir de confusion.

Le nom qualifié d'une classe interne est nom_classe_principale.nom_classe_interne. C'est donc le même principe que celui utilisé pour qualifier une classe contenue dans un package. La notation avec le point est donc légèrement étendue.

L'accès au membre de la classe principale est possible car le compilateur modifie le code de la classe principale et celui de la classe interne pour fournir à la classe interne une référence sur la classe principale.

Le code de la classe interne est modifié pour :

- ajouter une variable privée finale du type de la classe principale nommée this$0
- ajouter un paramètre supplémentaire dans le constructeur qui sera la classe principale et qui va initialiser la variable this$0

- utiliser cette variable pour préfixer les attributs de la classe principale utilisés dans la classe interne.

La code de la classe principale est modifié pour :

- ajouter une méthode static pour chaque champ de la classe principale qui attend en paramètre un objet de la classe principale. Cette méthode renvoie simplement la valeur du champ. Le nom de cette méthode est de la forme access$0
- modifier le code d'instanciation de la classe interne pour appeler le constructeur modifié

Dans le byte code généré, une variable privée finale contient une référence vers la classe principale. Cette variable est nommée this$0. Comme elle est générée par le compilateur, cette variable n'est pas utilisable dans le code source. C'est à partir de cette référence que le compilateur peut modifier le code pour accéder aux membres de la classe principale.

Pour pouvoir avoir acces aux membres de la classe principale, le compilateur génère dans la classe principale des accesseurs sur ces membres. Ainsi, dans la classe interne, pour accéder à un membre de la classe principale, le compilateur appelle un de ces accesseurs en utilisant la référence stockée. Ces méthodes ont un nom de la forme access$numero_unique et sont bien sûr inutilisables dans le code source puisqu'elles sont générées par le compilateur.

En tant que membre de la classe principale, une classe interne peut être déclarée avec le modificateur private ou protected.

Une classe peut faire référence dans le code source à son unique instance lors de l'exécution via le mot clé this. Une classe interne possède au moins deux références :

- l'instance de la classe interne elle même
- éventuellement les instances des classes internes dans laquelle la classe interne est imbriquée
- l'instance de sa classe principale

Dans la classe interne, il est possible pour accéder à une de ces instances d'utiliser le mot clé this préfixé par le nom de la classe suivi d'un point :
nom_classe_principale.this
nom_classe_interne.this

Le mot this seul désigne toujours l'instance de la classe courante dans son code source, donc this seul dans une classe interne désigne l'instance de cette classe interne.

Une classe interne non statique doit toujours être instanciée relativement à un objet implicite ou explicite du type de la classe principale. A la compilation, le compilateur ajoute dans la classe interne une référence vers la classe principale contenu dans une variable privée nommée this$0. Cette référence est initialisée avec un paramètre fourni au constructeur de la classe interne. Ce mécanisme permet de lier les deux instances.

La création d'une classe interne nécessite donc obligatoirement une instance de sa classe principale. Si cette instance n'est pas accessible, il faut en créer une et utiliser une notation particulière de l'opérateur new pour pouvoir instancier la classe interne. Par défaut, lors de l'instanciation d'une classe interne, si aucune instance de la classe principale n'est utilisée, c'est l'instance courante qui est utilisée (mot clé this).

Exemple :

```
public class ClassePrincipale14 {
        class ClasseInterne {
        }

        ClasseInterne ci = this. new ClasseInterne();
}
```

Pour créer une instance d'une classe interne dans une méthode statique de la classe principale, (la méthode main() par exemple), il faut obligatoirement instancier un objet de la classe principale avant et utiliser cet objet lors de la création de l'instance de la classe interne. Pour créer l'instance de la classe interne, il faut alors utiliser une syntaxe particulière de l'opérateur new.

Exemple :

```
public class ClassePrincipale15 {
        class ClasseInterne {
        }

        ClasseInterne ci = this. new ClasseInterne();

        static void maMethode() {
                ClassePrincipale15 cp = new ClassePrincipale15();
                ClasseInterne ci = cp. new ClasseInterne();
        }
}
```

Il est possible d'utiliser une syntaxe condensée pour créer les deux instances en une seule et même ligne de code.

Exemple :

```
public class ClassePrincipale19 {
        class ClasseInterne {
        }

        static void maMethode() {
                ClasseInterne ci = new ClassePrincipale19(). new ClasseInterne();
        }
}
```

Une classe peut hériter d'une classe interne. Dans ce cas, il faut obligatoirement fournir aux contructeurs de la classe une référence sur la classe principale de la classe mère et appeler explicitement dans le constructeur le constructeur de cette classe principale avec une notation particulière du mot clé super

Exemple :

```
public class ClassePrincipale9 {
        public class ClasseInterne {
        }

        class ClasseFille extends ClassePrincipale9.ClasseInterne {
                ClasseFille(ClassePrincipale9 cp) {
                        cp. super();
                }
        }
}
```

Une classe interne peut être déclarée avec les modificateurs final et abstract. Avec le modificateur final, la classe interne ne pourra être utilisée comme classe mère. Avec le modificateur abstract, la classe interne devra être étendue pour pouvoir être instanciée.

4.8.2. Les classes internes locales

Ces classes internes locales (local inner−classes) sont définies à l'intérieure d'une méthode ou d'un bloc de code. Ces classes ne sont utilisables que dans le bloc de code où elles sont définies. Les classes internes locales ont toujours accès aux membres de la classe englobante.

Exemple :

```
public class ClassePrincipale21 {
   int varInstance = 1;

   public static void main(String args[]) {
     ClassePrincipale21 cp = new ClassePrincipale21();
     cp.maMethode();
   }
```

```
   public void maMethode() {

     class ClasseInterne {
       public void affiche() {
         System.out.println("varInstance = " + varInstance);
       }
     }

     ClasseInterne ci = new ClasseInterne();
     ci.affiche();
   }
 }
```

Resultat :

```
C:\testinterne>javac ClassePrincipale21.java

C:\testinterne>java ClassePrincipale21
varInstance = 1
```

Leur particularité, en plus d'avoir un accès aux membres de la classe principale, est d'avoir aussi un accès à certaines variables locales du bloc ou est définie la classe interne.

Ces variables définies dans la méthode (variables ou paramètres de la méthode) sont celles qui le sont avec le mot clé final. Ces variables doivent être initialisée avant leur utilisation par la classe interne. Celles ci sont utilisables n'importe ou dans le code de la classe interne.

Le modificateur final désigne une variable dont la valeur ne peut être changée une fois qu'elle a été initialisée.

Exemple :

```
public class ClassePrincipale12 {

   public static void main(String args[]) {
     ClassePrincipale12 cp = new ClassePrincipale12();
     cp.maMethode();
   }

   public void maMethode() {
        int varLocale = 3;

     class ClasseInterne {
       public void affiche() {
         System.out.println("varLocale = " + varLocale);
       }
     }

     ClasseInterne ci = new ClasseInterne();
     ci.affiche();
   }
 }
```

Resultat :

```
javac ClassePrincipale12.java
ClassePrincipale12.java:14: Attempt to use a non-final variable varLocale from a
 different method. From enclosing blocks, only final local variables are availab
le.
         System.out.println("varLocale = " + varLocale);
                                              ^
1 error
```

Cette restriction est imposée par la gestion du cycle de vie d'une variable locale. Une telle variable n'existe que durant l'exécution de cette méthode. Une variable finale est une variable dont la valeur ne peut être modifiée après son initialisation. Ainsi, il est possible sans risque pour le compilateur d'ajouter un membre dans la classe interne et de copier

le contenu de la variable finale dedans.

Exemple :

```
public class ClassePrincipale13 {

   public static void main(String args[]) {
     ClassePrincipale13 cp = new ClassePrincipale13();
     cp.maMethode();
   }

   public void maMethode() {
        final int varLocale = 3;

     class ClasseInterne {
       public void affiche(final int varParam) {
         System.out.println("varLocale = " + varLocale);
         System.out.println("varParam  = " + varParam);
       }
     }

     ClasseInterne ci = new ClasseInterne();
     ci.affiche(5);
   }
 }
```

Resultat :

```
C:\>javac ClassePrincipale13.java

C:\>java ClassePrincipale13
varLocale = 3
varParam  = 5
```

Pour permettre à une classe interne locale d'accéder à une variable locale utilisée dans le bloc de code ou est définie la classe interne, la variable doit être stockée dans un endroit ou la classe interne pourra y accéder. Pour que cela fonctionne, le compilateur ajoute les variables nécessaires dans le constructeur de la classe interne.

Les variables accédées sont dupliquées dans la classe interne par le compilateur. Il ajoute pour chaque variable un membre privé dans la classe interne dont le nom est de la forme val$nom_variable. Comme la variable accédée est déclarée finale, cette copie peut être faite sans risque. La valeur de chacune de ces variables est fournie en paramètre du constructeur qui a été modifié par le compilateur.

Une classe qui est définie dans un bloc de code n'est pas un membre de la classe englobante : elle n'est donc pas accessible en dehors du bloc de code ou elle est définie. Ces restrictions sont équivalentes à la déclaration d'une variable dans un bloc de code.

Les variables ajoutées par le compilateur sont préfixées par this$ et val$. Ces variables et le constructeur modifié par le compilateur ne sont pas utilisables dans le code source.

Etant visible uniquement dans le bloc de code qui la définie, une classe interne locale ne peut pas utiliser les modificateurs public, private, protected et static dans sa définition. Leur utilisation provoque une erreur à la compilation.

Exemple :

```
public class ClassePrincipale11 {
        public void maMethode() {
                public class ClasseInterne {
                }
        }
}
```

Resultat :

```
javac ClassePrincipale11.java
ClassePrincipale11.java:2: '}' expected.
```

```
        public void maMethode() {
                                    ^
ClassePrincipale11.java:3: Statement expected.
                public class ClasseInterne {
                ^
ClassePrincipale11.java:7: Class or interface declaration expected.
}
^
3 errors
```

4.8.3. Les classes internes anonymes

Les classes internes anonymes (anonymous inner−classes) sont des classes internes qui ne possèdent pas de nom. Elles ne peuvent donc être instanciées qu'à l'endroit ou elles sont définies.

Ce type de classe est très pratique lorsqu'une classe doit être utilisée une seule fois : c'est par exemple le cas d'une classe qui doit être utilisée comme un callback.

Une syntaxe particulière de l'opérateur new permet de déclarer et instancier une classe interne :
new classe_ou_interface () {
// définition des attributs et des méthodes de la classe interne
}

Cette syntaxe particulière utilise le mot clé new suivi d'un nom de classe ou interface que la classe interne va respectivement étendre ou implémenter. La définition de la classe suit entre deux accolades. Une classe interne anonyme peut soit hériter d'une classe soit implémenter une interface mais elle ne peut pas explicitement faire les deux.

Si la classe interne étend une classe, il est possible de fournir des paramètres entre les parenthèses qui suivent le nom de la classe. Ces arguments éventuels fournis au moment de l'utilisation de l'opérateur new sont passés au constructeur de la super classe. En effet, comme la classe ne possède pas de nom, elle ne possède pas non plus de constructeur.

Les classes internes anonymes qui implémentent une interface héritent obligatoirement de classe Object. Comme cette classe ne possèdent qu'un constructeur sans paramètre, il n'est pas possible lors de l'instanciation de la classe interne de lui fournir des paramètres.

Une classe interne anonyme ne peut pas avoir de constructeur puisqu'elle ne possède pas de nom mais elle peut avoir des initialisateurs.

Exemple :

```
public void init() {
   boutonQuitter.addActionListener(
     new ActionListener() {
       public void actionPerformed(ActionEvent e) {
         System.exit(0);
       }
     }
   );
 }
```

Les classes anonymes sont un moyen pratique de déclarer un objet sans avoir a lui trouver un nom. La contre partie est que cette classe ne pourra être instanciée dans le code qu'à l'endroit ou elle est définie : elle est déclarée et instanciée en un seul et unique endroit.

Le compilateur génère un fichier ayant pour nom la forme suivante : nom_classe_principale$numéro_unique. En fait, le compilateur attribut un numéro unique à chaque classe interne anonyme et c'est ce numéro qui est donné au nom du fichier préfixé par le nom de la classe englobante et d'un signe '$'.

4.8.4. Les classes internes statiques

Les classes internes statiques (static member inner−classes) sont des classes internes qui ne possèdent pas de référence vers leur classe principale. Elles ne peuvent donc pas accéder aux membres d'instance de leur classe englobante. Elles peuvent toutefois avoir accès aux variables statiques de la classe englobante.

Pour les déclarer, il suffit d'utiliser en plus le modificateur static dans la déclaration de la classe interne.

Leur utilisation est obligatoire si la classe est utilisée dans une méthode statique qui par définition peut être appelée dans avoir d'instance de la classe et que l'on ne peut pas avoir une instance de la classe englobante. Dans le cas contraire, le compilateur indiquera une erreur :

Exemple :

```
public class ClassePrincipale4 {

        class ClasseInterne {
                public void afficher() {
                        System.out.println("bonjour");
                }
        }

        public static void main(String[] args) {
                new ClasseInterne().afficher();
        }
}
```

Resultat :

```
javac ClassePrincipale4.java
ClassePrincipale4.java:10: No enclosing instance of class ClassePrincipale4 is i
n scope; an explicit one must be provided when creating inner class ClassePrinci
pale4. ClasseInterne, as in "outer. new Inner()" or "outer. super()".
                new ClasseInterne().afficher();
                ^
1 error
```

En déclarant la classe interne static, le code se compile et peut être exécuté

Exemple :

```
public class ClassePrincipale4 {

        static class ClasseInterne {
                public void afficher() {
                        System.out.println("bonjour");
                }
        }

        public static void main(String[] args) {
                new ClasseInterne().afficher();
        }
}
```

Resultat :

```
javac ClassePrincipale4.java

java ClassePrincipale4
bonjour
```

Comme elle ne possède pas de référence sur sa classe englobante, une classe interne statique est en fait traduite par le compilateur comme une classe principale. En fait, il est difficile de les mettre dans une catégorie (classe principale ou classe interne) car dans le code source c'est une classe interne (classe définie dans une autre) et dans le byte code généré c'est une classe principale.

Ce type de classe n'est pas très employé.

4.9. La gestion dynamique des objets

Tout objet appartient à une classe et Java sait la reconnaitre dynamiquement.

Java fournit dans son API un ensemble de classes qui permettent d'agir dynamiquement sur des classes. Cette technique est appellée introspection et permet :

- de décrire une classe ou une interface : obtenir son nom, sa classe mère, la liste de ses méthodes, de ses variables de classe, de ses constructeurs et de ses variables d'instances
- d'agir sur une classe en envoyant, à un objet Class des message comme à tout autre objet. Par exemple, créer dynamiquement à partir d'un objet Class une nouvelle instance de la classe représentée

Voir le chapitre sur la gestion dynamique des objets et l'introspection pour obtenir plus d'informations.

5. Les packages de bases

Le JDK se compose de nombreuses classes regroupées selon leur fonctionnalité en packages. La première version du JDK était composée de 8 packages qui constituent encore aujourd'hui les packages de bases des différentes versions du JDK.

Ce chapitre contient plusieurs sections :

5.1. Liste des packages selon la version du JDK

Selon sa version, JDK contient un certain nombre de packages, chacun étant constitué par un ensemble de classes qui couvrent un même domaine et apportent de nombreuses fonctionnalités. Les différentes versions du JDK sont constamment enrichis avec de nouveaux packages :

Packages	JDK 1.0	JDK 1.1	JDK 1.2	JDK 1.3	JDK 1.4	JDK 1.5
java.applet Développement des applets	X	X	X	X	X	
java.awt Toolkit pour interfaces graphiques	X	X	X	X	X	
java.awt.color Gérer et utiliser les couleurs			X	X	X	
java.awt.datatransfer Echanger des données via le presse−papier		X	X	X	X	
java.awt.dnd Gérer le cliquer/glisser			X	X	X	
java.awt.event Gérer les évenements utilisateurs		X	X	X	X	
java.awt.font Uiliser les fontes			X	X	X	
java.awt.geom dessiner des formes géométriques			X	X	X	
java.awt.im				X	X	

java.awt.im.spi				X	X	
java.awt.image Afficher des images		X	X	X	X	
java.awt.image.renderable Modifier le rendu des images			X	X	X	
java.awt.print Réaliser des impressions			X	X	X	
java.beans Développer des composants réutilisables		X	X	X	X	
java.beans.beancontext			X	X	X	
java.io Gérer les flux	X	X	X	X	X	
java.lang Classes de base du langage	X	X	X	X	X	
java.lang.ref			X	X	X	
java.lang.reflect Utiliser la réflexion (introspection)		X	X	X	X	
java.math Utiliser des opérations mathématiques		X	X	X	X	
java.net Utiliser les fonctionnalités réseaux	X	X	X	X	X	
java.nio					X	
java.nio.channels					X	
java.nio.channels.spi					X	
java.nio.charset					X	
java.nio.charset.spi					X	
java.rmi Développement d'objets distribués		X	X	X	X	
java.rmi.activation			X	X	X	
java.rmi.dgc		X	X	X	X	
java.rmi.registry		X	X	X	X	
java.rmi.server Gérer les objets serveurs de RMI		X	X	X	X	
java.security Gérer les signatures et les certifications		X	X	X	X	
java.security.acl		X	X	X	X	
java.security.cert		X	X	X	X	
java.security.interfaces		X	X	X	X	
java.security.spec			X	X	X	
java.sql JDBC pour l'accès aux bases de données		X	X	X	X	
java.text Formatter des objets en texte		X	X	X	X	
	X	X	X	X	X	

java.util Utilitaires divers						
java.util.jar Gérer les fichiers jar			X	X	X	
java.util.logging Utiliser des logs					X	
java.util.prefs Gérer des préférences					X	
java.util.regex Utiliser les expressions régulières					X	
java.util.zip Gérer les fichiers zip		X	X	X	X	
javax.accessibility			X	X	X	
javax.crypto Utiliser le cryptage des données					X	
javax.crypto.interfaces					X	
javax.crypto.spec					X	
javax.imageio					X	
javax.imageio.event					X	
javax.imageio.metadata					X	
javax.imageio.plugins.jpeg					X	
javax.imageio.spi					X	
javax.imageio.stream					X	
javax.naming				X	X	
javax.naming.directory				X	X	
javax.naming.event				X	X	
javax.naming.ldap				X	X	
javax.naming.spi				X	X	
javax.net					X	
javax.net.ssl Utiliser une connexion réseau securisée avec SSL					X	
javax.print					X	
javax.print.attribute					X	
javax.print.attribute.standard					X	
javax.print.event					X	
javax.rmi				X	X	
javax.rmi.CORBA				X	X	
javax.security.auth API JAAS pour l'authentification et l'autorisation					X	
javax.security.auth.callback					X	
javax.security.auth.kerberos					X	
javax.security.auth.login					X	

javax.security.auth.spi					X	
javax.security.auth.x500					X	
javax.security.cert					X	
javax.sound.midi				X	X	
javax.sound.midi.spi				X	X	
javax.sound.sampled				X	X	
javax.sound.sampled.spi				X	X	
javax.sql					X	
javax.swing Swing pour développer des interfaces graphiques			X	X	X	
javax.swing.border Gérer les bordures des composants Swing			X	X	X	
javax.swing.colorchooser Composant pour sélectionner une couleur			X	X	X	
javax.swing.event Gérer des évenements utilisateur des composants Swing			X	X	X	
javax.swing.filechooser Composant pour sélectionner un fichier			X	X	X	
javax.swing.plaf Gérer l'aspect des composants Swing			X	X	X	
javax.swing.plaf.basic			X	X	X	
javax.swing.plaf.metal Gérer l'aspect metal des composants Swing			X	X	X	
javax.swing.plaf.multi			X	X	X	
javax.swing.table			X	X	X	
javax.swing.text			X	X	X	
javax.swing.text.html			X	X	X	
javax.swing.text.html.parser			X	X	X	
javax.swing.text.rtf			X	X	X	
javax.swing.tree Un composant de type arbre			X	X	X	
javax.swing.undo Gérer les annulations d'opérations d'édition			X	X	X	
javax.transaction				X	X	
javax.transaction.xa					X	
javax.xml.parsers API JAXP pour utiliser XML					X	
javax.xml.transform transformer un document XML avec XSLT					X	
javax.xml.transform.dom					X	
javax.xml.transform.sax					X	
javax.xml.transform.stream					X	

org.ietf.jgss					X	
org.omg.CORBA			X	X	X	
org.omg.CORBA_2_3				X	X	
org.omg.CORBA_2_3.portable				X	X	
org.omg.CORBA.DynAnyPackage			X	X	X	
org.omg.CORBA.ORBPackage			X	X	X	
org.omg.CORBA.portable			X	X	X	
org.omg.CORBA.TypeCodePackage			X	X	X	
org.omg.CosNaming			X	X	X	
org.omg.CosNaming.NamingContextExtPackage			X	X	X	
org.omg.CosNaming.NamingContextPackage					X	
org.omg.Dynamic					X	
org.omg.DynamicAny					X	
org.omg.DynamicAny.DynAnyFactoryPackage					X	
org.omg.DynamicAny.DynAnyPackage					X	
org.omg.IOP					X	
org.omg.IOP.CodecFactoryPackage					X	
org.omg.IOP.CodecPackage					X	
org.omg.Messaging					X	
org.omg.PortableInterceptor					X	
org.omg.PortableInterceptor.ORBInitInfoPackage					X	
org.omg.PortableServer					X	
org.omg.PortableServer.CurrentPackage					X	
org.omg.PortableServer.POAPackage					X	
org.omg.PortableServer.ServantLocatorPackage					X	
org.omg.PortableServer.portable					X	
org.omg.SendingContext				X	X	
org.omg.stub.java.rmi				X	X	
org.w3c.dom Utiliser DOM pour un document XML					X	
org.xml.sax Utiliser SAX pour un document XML					X	
org.xml.sax.ext					X	
org.xml.sax.helpers					X	

5.2. Le package java.lang

Ce package de base contient les classes fondamentales tel que Object, Class, Math, System, String, StringBuffer, Thread, les wrapper etc ... Certaines de ces classes sont détaillées dans les sections suivantes.

Il contient également plusieurs classes qui permettent de demander des actions au système d'exploitation sur laquelle la machine virtuelle tourne, par exemple les classes ClassLoader, Runtime, SecurityManager.

Certaines classes sont détaillées dans des chapitres dédiés : la classe Math est détaillée dans le chapitre "les fonctions mathématiques", l classe Class est détaillée dans le chapitre "la gestion dynamique des objets et l'introspection" et la classe Thread est détaillée dans le chapitre "le multitache".

Ce package est tellement fondamental qu'il est implicitement importé dans tous les fichiers sources par le compilateur.

5.2.1. La classe Object

C'est la super classe de toutes les classes Java : toutes ces méthodes sont donc héritées par toutes les classes.

5.2.1.1. La méthode getClass()

La méthode getClass() renvoie un objet de la classe Class qui représente la classe de l'objet.

Le code suivant permet de connaitre le nom de la classe de l'objet

Exemple :

```
String nomClasse = monObject.getClass().getName();
```

5.2.1.2. La méthode toString()

La méthode toString() de la classe Object renvoie le nom de la classe , suivi du séparateur @, lui même suivi par la valeur de hachage de l'objet.

5.2.1.3. La méthode equals()

La méthode equals() implémente une comparaison par défaut. Sa définition dans Object compare les références : donc obj1.equals(obj2) ne renverra true que si obj1 et obj2 désignent le même objet. Dans une sous classe de Object, pour laquelle on a besoin de pouvoir dire que deux objets distincts peuvent être égaux, il faut redéfinir la méthode equals héritée de Object.

5.2.1.4. La méthode finalize()

A l'inverse de nombreux langages orientés objet tel que le C++ ou Delphi, le programmeur Java n'a pas à se préoccuper de la destruction des objets qu'il instancie. Ceux−ci sont détruits et leur emplacement mémoire est récupéré par le ramasse miette de la machine virtuelle dès qu'il n'y a plus de référence sur l'objet.

La machine virtuelle garantit que toutes les ressources Java sont correctement libérées mais, quand un objet encapsule une ressource indépendante de Java (comme un fichier par exemple), il peut être préférable de s'assurer que la ressource sera libérée quand l'objet sera détruit. Pour cela, la classe Object définit la méthode protected finalize, qui est appelée quand le ramasse miettes doit récupérer l'emplacement de l'objet ou quand la machine virtuelle termine son exécution

Exemple :

```
import java.io.*;

public class AccesFichier {
   private FileWriter fichier;

   public AccesFichier(String s) {
```

```
      try {
      fichier = new FileWriter(s);
      }
      catch (IOException e) {
         System.out.println("Impossible d'ouvrir le fichier
```

5.2.1.5. La méthode clone()

Si x désigne un objet obj1, l'exécution de x.clone() renvoie un second objet obj2, qui est une copie de obj1 : si obj1 est ensuite modifié, obj2 n'est pas affecté par ce changement.

Par défaut, la méthode clone(), héritée de Object fait une copie variable par variable : elle offre donc un comportement acceptable pour de très nombreuses sous classe de Object. Cependant comme le processus de duplication peut être délicat à gérer pour certaines classes (par exemple des objets de la classe Container), l'héritage de clone ne suffit pas pour qu'une classe supporte le clonage.

Pour permettre le clonage d'une classe, il faut implémenter dans la classe l'interface Cloneable.

La première chose que fait la méthode clone() de la classe Object, quand elle est appelée, est de tester si la classe implémente Cloneable. Si ce n'est pas le cas, elle lève l'exception CloneNotSupportedException.

5.2.2. La classe String

Une chaîne de caractères est contenue dans un objet de la classe String

On peut initialiser une variable String sans appeler explicitement un contructeur : le compilateur se charge de créer un objet.

Exemple : deux déclarations de chaines identiques.

```
String uneChaine = «bonjour»;
String uneChaine = new String(«bonjour»);
```

Les objets de cette classe ont la particularité d'être constants. Chaque traitement qui vise à transformer un objet de la classe est implémenté par une méthode qui laisse l'objet d'origine inchangé et renvoie un nouvel objet String contenant les modifications.

Exemple :

```
private String uneChaine;
void miseEnMajuscule(String chaine) {
    uneChaine = chaine.toUpperCase()
}
```

Il est ainsi possible d'enchainer plusieurs méthodes :

Exemple :

```
uneChaine = chaine.toUpperCase().trim();
```

L'opérateur + permet la concatenation de chaines de caractères.

La comparaison de deux chaînes doit se faire via la méthode equals() qui compare les objets eux même et non l'opérateur == qui compare les réferences de ces objets :

Exemple :

```
String nom1 = new String("Bonjour");
String nom2 = new String("Bonjour");
System.out.println(nom1 == nom2); // affiche false
System.out.println( nom1.equals(nom2)); // affiche true
```

Cependant dans un souci d'efficacité, le compilateur ne duplique pas 2 constantes chaines de caractères : il optimise l'espace mémoire utilisé en utilisant le même objet. Cependant, l'appel explicite du constructeur ordonne au compilateur de créer un nouvel objet.

Exemple :

```
String nom1 = «Bonjour»;
String nom2 = «Bonjour»;
String nom3 = new String(«Bonjour»);
System.out.println(nom1 == nom2); // affiche true
System.out.println(nom1 == nom3); // affiche false
```

La classe String possède de nombreuses méthodes dont voici les principales :

Méthodes la classe String	Role
charAt(int)	renvoie le nieme caractère de la chaine
compareTo(String)	compare la chaîne avec l'argument
concat(String)	ajoute l'argument à la chaîne et renvoie la nouvelle chaîne
endsWith(String)	vérifie si la chaîne se termine par l'argument
equalsIgnoreCase(String)	compare la chaîne sans tenir compte de la casse
indexOf(String)	renvoie la position de début à laquelle l'argument est contenu dans la chaine
lastIndexOf(String)	renvoie la dernière position à laquelle l'argument est contenu dans la chaine
lenght()	renvoie la longueur de la chaine
replace(char,char)	renvoie la chaîne dont les occurences d'un caractère sont remplacées
startsWidt(String int)	Vérifie si la chaîne commence par la sous chaîne
substring(int,int)	renvoie une partie de la chaine
toLowCase()	renvoie la chaîne en minuscule
toUpperCase()	renvoie la chaîne en majuscule
trim()	enlève les caractères non significatifs de la chaine

5.2.3. La classe StringBuffer

Les objets de cette classe contiennent des chaines de caractères variables, ce qui permet de les agrandir ou de les réduire. Cette objet peut être utilisé pour construire ou modifier une chaîne de caractères chaque fois que l'utilisation de la classe String nécessiterait de nombreuses instanciations d'objets temporaires.

Par exemple, si str est un objet de type String, le compilateur utilisera la classe StringBuffer pour traiter la concaténation de « abcde »+str+« z » en générant le code suivant : new StringBuffer().append(« abcde »).append(str).append(« z »).toString();

Ce traitement aurait pu être réalisé avec trois appels à la méthode concat() de la classe String mais chacun des appels aurait instancié un objet StringBuffer pour réaliser la concaténation, ce qui est couteux en temps exécution

La classe StringBuffer dispose de nombreuses méthodes qui permettent de modifier le contenu de la chaîne de caractère

Exemple (code Java 1.1) : une méthode uniquement pédagogique

```
public class MettreMaj {

   static final String lMaj = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   static final String lMin = "abcdefghijklmnopqrstuvwxyz";

   public static void main(java.lang.String[] args) {
      System.out.println(MetMaj("chaine avec MAJ et des min"));
   }

   public static String MetMaj(String s) {
      StringBuffer sb = new StringBuffer(s);

      for ( int i = 0; i <sb.length(); i++) {
          int index = lMin.indexOf(sb.charAt(i));
          if (index >=0 ) sb.setCharAt(i,lMaj.charAt(index));
      }
      return sb.toString();
   }
}
```

Résultat :

```
CHAINE AVEC MAJ ET DES MIN
```

5.2.4. Les wrappers

Les objet de type wrappers (enveloppeurs) représentent des objets qui encapsulent une donnée de type primitif et qui fournissent un ensemble de méthodes qui permettent notamment de faire des conversions.

Ces classes offrent toutes les services suivants :

- un constructeur qui permet une instanciation à partir du type primitif et un constructeur qui permet une instanciation à partir d'un objet String
- une méthode pour fournir la valeur primitive représentée par l'objet
- une méthode equals() pour la comparaison.

Les méthodes de conversion opèrent sur des instances, mais il est possible d'utiliser des méthodes statiques.

Exemple :

```
int valeur =
Integer.valueOf("999").intValue();
```

Ces classes ne sont pas interchangeables avec les types primitifs d'origine car il s'agit d'objet.

Exemple :

```
Float objetpi = new Float(«3.1415»);

System.out.println(5*objetpi); // erreur à la compil
```

Pour obtenir la valeur contenue dans l'objet, il faut utiliser la méthode typeValue() ou type est le nom du type standard

Exemple :

```
Integer Entier = new Integer(«10»);

int entier = Entier.intValue();
```

Les classes Integer, Long, Float et Double définissent toutes les constantes MAX_VALUE et MIN_VALUE qui représentent leurs valeurs minimales et maximales.

Lorsque l'on effectue certaines opérations mathématiques sur des nombres à virgules flottantes (float ou double), le résultat peut prendre l'une des valeurs suivantes :

- NEGATIVE_INFINITY : infini négatif causé par la division d'un nombre négatif par 0.0
- POSITIVE_INFINITY : infini positif causé par la division d'un nombre positif par 0.0
- NaN: n'est pas un nombre (Not a Number) causé par la division de 0.0 par 0.0

Il existe des méthodes pour tester le résultat :

Float.isNaN(float); //idem avec double
Double.isInfinite(double); // idem avec float

Exemple :

```
float res = 5.0f / 0.0f;

if (Float.isInfinite(res)) { ... };
```

La constante Float.NaN n'est ni égale à un nombre dont la valeur est NaN ni à elle même. Float.NaN == Float.NaN retourne False

Lors de la division par zéro d'un nombre entier, une exception est levée.

Exemple :

```
System.out.println(10/0);

Exception in thread "main" java.lang.ArithmeticException: / by zero
at test9.main(test9.java:6)
```

5.2.5. La classe System

Cette classe possède de nombreuses fonctionnalités pour utiliser des services du système d'exploitation.

5.2.5.1. L'utilisation des flux d'entrée/sortie standard

La classe System défini trois variables statiques qui permettent d'utiliser les flux d'entrée/sortie standards du système d'exploitation.

Variable	Type	Rôle
in	PrintStream	Entrée standard du système. Par défaut, c'est le clavier.
out	InputStream	Sortie standard du système. Par défaut, c'est le moniteur.
err	PrintStream	Sortie standard des erreurs du système. Par défaut, c'est le moniteur.

Exemple :

```
System.out.println("bonjour");
```

La classe système possède trois méthodes qui permettent de rediriger ces flux.

La méthode printf() dont le mode fonctionnement bien connu dans le langage C a été reprise pour être ajoutée dans l'API de Java.

Exemple (java 1.5):

```
public class TestPrintf {
  public static void main(String[] args) {
    System.out.printf("%4d",32);
  }
}
```

La méthode printf propose :

- un nombre d'arguments variable
- des formats standard pour les types primitifs, String et Date
- des justifications possibles avec certains formats
- l'utilisation de la localisation pour les données numériques et de type date

Exemple (java 1.5):

```
import java.util.*;

public class TestPrintf2 {

  public static void main(String[] args) {
    System.out.printf("%d \n"                      ,13);
    System.out.printf("%4d \n"                     ,13);
    System.out.printf("%04d \n"                    ,13);
    System.out.printf("%f \n"                      ,3.14116);
    System.out.printf("%.2f \n"                    ,3.14116);
    System.out.printf("%s \n"                      ,"Test");
    System.out.printf("%10s \n"                    ,"Test");
    System.out.printf("%-10s \n"                   ,"Test");
    System.out.printf("%tD \n"                     , new Date());
    System.out.printf("%tF \n"                     , new Date());
    System.out.printf("%1$te %1$tb %1$ty \n"       , new Date());
    System.out.printf("%1$tA %1$te %1$tB %1$tY \n", new Date());
    System.out.printf("%1$tr \n"                   , new Date());
  }

}
```

Résultat :

```
C:\tiger>java TestPrintf2
13
13
0013
3,141160
3,14
Test
Test
Test
08/23/04
2004-08-23
23 ao¹t 04
lundi 23 ao¹t 2004
03:56:25 PM
```

Une exception est levée lors de l'exécution si un des formats utilisés est inconnu

Exemple (java 1.5) :

```
C:\tiger>java TestPrintf2
13 1300133,1411603,14Test TestTest 08/23/04Exception in thread "main"
```

```
java.util.UnknownFormatConversionException: Conversion = 'tf'
at java.util.Formatter$FormatSpecifier.checkDateTime(Unknown Source)
at java.util.Formatter$FormatSpecifier.<init>(Unknown Source)
at java.util.Formatter.parse(Unknown Source)
at java.util.Formatter.format(Unknown Source)
at java.io.PrintStream.format(Unknown Source)
at java.io.PrintStream.printf(Unknown Source)
at TestPrintf2.main(TestPrintf2.java:15)
```

5.2.5.2. Les variables d'environnement et les propriétés du système

JDK 1.0 propose la méthode statique getEnv() qui renvoie la valeur de la propriété système dont le nom est fourni en paramètre.

Depuis le JDK 1.1, cette méthode est deprecated car elle n'est pas multi−système. Elle est remplacée par un autre mécanisme qui n'interroge pas directement le système mais qui recherche les valeurs dans un ensemble de propriétés. Cet ensemble est constitué de propriétés standard fournies par l'environnement java et par des propriétés ajoutées par l'utilisateur.

Voici une liste non exhaustive des propriétés fournies par l'environnement java :

Nom de la propriété	Rôle
java.version	Version du JRE
java.vendor	Auteur du JRE
java.vendor.url	URL de l'auteur
java.home	Répertoire d'installation de java
java.vm.version	Version de l'implémentation la JVM
java.vm.vendor	Auteur de l'implémentation de la JVM
java.vm.name	Nom de l'implémentation de la JVM
java.specification.version	Version des spécifications de la JVM
java.specification.vendor	Auteur des spécifications de la JVM
java.specification.name	Nom des spécifications de la JVM
java.ext.dirs	Chemin du ou des répertoires d'extension
os.name	Nom du système d'exploitation
os.arch	Architecture du système d'exploitation
os.version	Version du système d'exploitation
file.separator	Séparateur de fichiers (exemple : "/" sous Unix, "\" sous Windows)
path.separator	Séparateur de chemin (exemple : ":" sous Unix, ";" sous Windows)
line.separator	Séparateur de ligne
user.name	Nom du user courant
user.home	Répertoire d'accueil du user courant
user.dir	Répertoire courant au moment de l'initialisation de la propriété

Pour définir ces propres propriétés, il faut utiliser l'option −D de l'interpreteur java en utilisant la ligne de commande.

La méthode statique getProperty() permet d'obtenir la valeur de la propriété dont le nom est fourni en paramètre. Une version surchargée de cette méthode permet de préciser un second paramètre qui contiendra la valeur par défaut, si la propriété n'est pas définie.

Exemple :

```
public class TestProperty {

  public static void main(String[] args) {
    System.out.println("java.version                 ="+System.getProperty("java.version"));
    System.out.println("java.vendor                  ="+System.getProperty("java.vendor"));
    System.out.println("java.vendor.url              ="+System.getProperty("java.vendor.url"));
    System.out.println("java.home                    ="+System.getProperty("java.home"));
    System.out.println("java.vm.specification.version ="
      +System.getProperty("java.vm.specification.version"));
    System.out.println("java.vm.specification.vendor  ="
      +System.getProperty("java.vm.specification.vendor"));
    System.out.println("java.vm.specification.name    ="
      +System.getProperty("java.vm.specification.name"));
    System.out.println("java.vm.version              ="+System.getProperty("java.vm.version"));
    System.out.println("java.vm.vendor               ="+System.getProperty("java.vm.vendor"));
    System.out.println("java.vm.name                 ="+System.getProperty("java.vm.name"));
    System.out.println("java.specification.version    ="
      +System.getProperty("java.specification.version"));
    System.out.println("java.specification.vendor     ="
      +System.getProperty("java.specification.vendor"));
    System.out.println("java.specification.name       ="
      +System.getProperty("java.specification.name"));
    System.out.println("java.class.version           ="
      +System.getProperty("java.class.version"));
    System.out.println("java.class.path              ="
      +System.getProperty("java.class.path"));
    System.out.println("java.ext.dirs                ="+System.getProperty("java.ext.dirs"));
    System.out.println("os.name                      ="+System.getProperty("os.name"));
    System.out.println("os.arch                      ="+System.getProperty("os.arch"));
    System.out.println("os.version                   ="+System.getProperty("os.version"));
    System.out.println("file.separator               ="+System.getProperty("file.separator"));
    System.out.println("path.separator               ="+System.getProperty("path.separator"));
    System.out.println("line.separator               ="+System.getProperty("line.separator"));
    System.out.println("user.name                    ="+System.getProperty("user.name"));
    System.out.println("user.home                    ="+System.getProperty("user.home"));
    System.out.println("user.dir                     ="+System.getProperty("user.dir"));
  }
}
```

Par défaut, l'accès aux propriétés système est restreint par le SecurityManager pour les applets.

5.2.6. La classe Runtime

La classe Runtime permet d'intéragir avec le système dans lequel l'application s'exécute : obtenir des informations sur le système, arrêt de la machine virtuelle, éxecution d'un programme externe

Cette classe ne peut pas être instanciée mais il est possible d'obtenir une instance en appelant la méthode statique getRuntime().

La méthode exec() permet d'exécuter des commandes du système d'exploitation ou s'exécute la JVM. Elle lance la commande de manière asynchrone et renvoie un objet de type Process pour obtenir des informations sur le processus lancé.

L'inconvénient d'utiliser cette méthode est que la commande executée est dépendante du système d'exploitation.

La suite de cette section sera développée dans une version future de ce document

5.3. Présentation rapide du package awt java

AWT est une collection de classes pour la réalisation d'applications graphiques ou GUI (Graphic User Interface)

Les composants qui sont utilisés par les classes définies dans ce package sont des composants dit "lourds" : ils dépendent entièrement du système d'exploitation. D'ailleurs leur nombre est limité car ils sont communs à plusieurs systèmes d'exploitation pour assurer la portabilité. Cependant, la représentation d'une interface graphique avec awt sur plusieurs systèmes peut ne pas être identique.

AWT se compose de plusieurs packages dont les principaux sont:

- java.awt : c'est le package de base de la bibliothèque AWT
- java.awt.images : ce package permet la gestion des images
- java.awt.event : ce package permet la gestion des événements utlisateurs
- java.awt.font : ce package permet d'utiliser les polices de caractères
- java.axt.dnd : ce package permet l'utilisation du cliquer/glisser

Le chapitre "Création d'interface graphique avec AWT" détaille l'utilisation de ce package,

5.4. Présentation rapide du package java.io

Ce package définit un ensemble de classes pour la gestion des flux d'entrées−sorties

Le chapitre les flux détaille l'utilisation de ce package

5.5. Le package java.util

Ce package contient un ensemble de classes utilitaires : la gestion des dates (Date et Calendar), la génération de nombres aléatoires (Random), la gestion des collections ordonnées ou non tel que la table de hachage (HashTable), le vecteur (Vector), la pile (Stack) ..., la gestion des propriétés (Properties), des classes dédiées à l'internationalisation (ResourceBundle, PropertyResourceBundle, ListResourceBundle) etc ...

Certaines de ces classes sont présentées plus en détail dans les sections suivantes.

5.5.1. La classe StringTokenizer

Cette classe permet de découper une chaîne de caractères (objet de type String) en fonction de séparateurs. Le constructeur de la classe accepte 2 paramètres : la chaîne à décomposer et une chaîne contenant les séparateurs

Exemple (code Java 1.1) :

```
import java.util.*;

class test9 {
   public static void main(String args[]) {
      StringTokenizer st = new StringTokenizer("chaine1,chaine2,chaine3,chaine4",",");
      while (st.hasMoreTokens()) {
         System.out.println((st.nextToken()).toString());
      }
```

```
   }
}

C:\java>java test9
chaine1
chaine2
chaine3
chaine4
```

La méthode hasMoreTokens() fournit un contrôle d'itération sur la collection en renvoyant un booleen indiquant si il reste encore des éléments.

La méthode getNextTokens() renvoie le prochain élément sous la forme d'un objet String

5.5.2. La classe Random

La classe Random permet de générer des nombres pseudo−aléatoires. Après l'appel au constructeur, il suffit d'appeler la méthode correspondant au type désiré : nextInt(), nextLong(), nextFloat() ou nextDouble()

Méthodes	valeur de retour
nextInt	entre Integer.MIN_VALUE et Interger.MAX_VALUE
nextLong	entre long.MIN_VALUE et long.MAX_VALUE
nextFloat ou nextDouble	entre 0.0 et 1.0

Exemple (code Java 1.1) :

```
import java.util.*;
class test9 {
   public static void main (String args[]) {
      Random r = new Random();
      int a = r.nextInt() %10; //entier entre −9 et 9
      System.out.println("a = "+a);
   }
}
```

5.5.3. Les classes Date et Calendar

En java 1.0, la classe Date permet de manipuler les dates.

Exemple (code Java 1.0) :

```
import java.util.*;
...
    Date maintenant = new Date();
    if (maintenant.getDay() == 1)
       System.out.println(" lundi ");
```

Le constructeur d'un objet Date l'initialise avec la date et l'heure courante du système.

Exemple (code Java 1.0) : recherche et affichage de l'heure

```
import java.util.*;
import java.text.*;

public class TestHeure {
   public static void main(java.lang.String[] args) {
      Date date = new Date();
```

```
      System.out.println(DateFormat.getTimeInstance().format(date));
   }
}
```

Résultat :

```
22:05:21
```

La méthode getTime() permet de calculer le nombre de millisecondes écoulées entre la date qui est encapsulée dans l'objet qui reçoit le message getTime et le premier janvier 1970 à minuit GMT.

En java 1.1, de nombreuses méthodes et constructeurs de la classe Date sont deprecated, notamment celles qui permettent de manipuler les éléments qui composent la date et leur formattage : il faut utiliser la classe Calendar.

Exemple (code Java 1.1) :

```
import java.util.*;

public class TestCalendar {
   public static void main(java.lang.String[] args) {

      Calendar c = Calendar.getInstance();
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.MONDAY)
          System.out.println(" nous sommes lundi ");
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.TUESDAY)
          System.out.println(" nous sommes mardi ");
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.WEDNESDAY)
          System.out.println(" nous sommes mercredi ");
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.THURSDAY)
          System.out.println(" nous sommes jeudi ");
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.FRIDAY)
          System.out.println(" nous sommes vendrei ");
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY)
          System.out.println(" nous sommes samedi ");
      if (c.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY)
          System.out.println(" nous sommes dimanche ");

   }
}
```

Résultat :

```
nous sommes lundi
```

5.5.4. La classe Vector

Un objet de la classe Vector peut être considéré comme une tableau évolué qui peut contenir un nombre indéterminé d'objets.

Les méthodes principales sont les suivantes :

Méthode	Rôle
void addElement(Object)	ajouter un objet dans le vecteur
boolean contains(Object)	retourne true si l'objet est dans le vecteur
Object elementAt(int)	retourne l'objet à l'index indiqué
Enumeration elements()	retourne une enumeration contenant tous les éléments du vecteur
Object firstElement()	

	retourne le premier élément du vecteur (celui dont l'index est égal à zéro)
int indexOf(Object)	renvoie le rang de l'élément ou −1
void insertElementAt(Object, int)	insérer un objet à l'index indiqué
boolean isEmpty()	retourne un booléen si le vecteur est vide
Objet lastElement()	retourne le dernier élément du vecteur
void removeAllElements()	vider le vecteur
void removeElement(Object)	supprime l'objet du vecteur
void removeElementAt(int)	supprime l'objet à l'index indiqué
void setElementAt(object, int)	remplacer l'élément à l'index par l'objet
int size()	nombre d'objet du vecteur

On peut stocker des objets de classes différentes dans un vecteur mais les éléments stockés doivent obligatoirement être des objets (pour le type primitif il faut utiliser les wrappers tel que Integer ou Float mais pas int ou float).

Exemple (code Java 1.1) :

```
Vector v = new Vector();
v.addElement(new Integer(10));
v.addElement(new Float(3.1416));
v.insertElementAt("chaine ",1);
System.out.println(" le vecteur contient "+v.size()+ " elements ");
String retrouve = (String) v.elementAt(1);
System.out.println(" le 1er element = "+retrouve);

C:\$user\java>java test9
le vecteur contient 3 elements
le 1er element = chaine
```

Exemple (code Java 1.1) : le parcours d'un vecteur

```
  Vector v = new Vector();
   ...

   for (int i = 0; i < v.size() ; i ++) {
      System.out.println(v.elementAt(i));
   }
```

Il est aussi possible de parcourir l'ensemble des éléments en utilisant une instance de l'interface Enumeration.

5.5.5. La classe Hashtable

Les informations d'une Hastable sont stockées sous la forme clé − données. Cet objet peut être considéré comme un dictionnaire.

Exemple (code Java 1.1) :

```
Hashtable dico = new Hashtable();
dico.put(«livre1», « titre du livre 1 »);
dico.put(«livre2», «titre du livre 2 »);
```

Il est possible d'utiliser n'importe quel objet comme clé et comme donnée

Exemple (code Java 1.1) :

```
dico.put(«jour», new Date());
dico.put(new Integer(1),«premier»);
dico.put(new Integer(2),«deuxième»);
```

Pour lire dans la table, on utilise get(object) en donnant la clé en paramètre.

Exemple (code Java 1.1) :

```
System.out.println(« nous sommes le » +dico.get(«jour»));
```

La méthode remove(Object) permet de supprimer une entrée du dictionnaire correspondant à la clé passée en paramètre.

La méthode size() permet de connaitre le nombre d'association du dictionnaire.

5.5.6. L'interface Enumeration

L'interface Enumeration est utilisée pour permettre le parcours séquentiel de collections.

Enumeration est une interface qui définit 2 méthodes :

Méthodes	Rôle
boolean hasMoreElements()	retourne true si l'énumeration contient encore un ou plusieurs elements
Object nextElement()	retourne l'objet suivant de l'énumération Elle leve une Exception NoSuchElementException si la fin de la collection est atteinte.

Exemple (code Java 1.1) : contenu d'un vecteur et liste des clés d'une Hastable

```
import java.util.*;

class test9 {

   public static void main (String args[]) {

      Hashtable h = new Hashtable();
      Vector v = new Vector();

      v.add("chaine 1");
      v.add("chaine 2");
      v.add("chaine 3");

      h.put("jour", new Date());
      h.put(new Integer(1),"premier");
      h.put(new Integer(2),"deuxième");

      System.out.println("Contenu du vector");

      for (Enumeration e = v.elements() ; e.hasMoreElements() ; ) {
         System.out.println(e.nextElement());
      }

      System.out.println("\nContenu de la hashtable");

      for (Enumeration e = h.keys() ; e.hasMoreElements() ; ) {
         System.out.println(e.nextElement());
      }
   }
}
```

```
C:\$user\java>java test9
Contenu du vector
chaine 1
chaine 2
chaine 3
Contenu de la hashtable
jour
2
1
```

5.5.7. Les expressions régulières

Le JDK 1.4 propose une api en standard pour utiliser les expressions régulières. Les expressions régulières permettent de comparer une chaîne de caractères àun motif pour vérifier qu'il y a concordance.

La package java.util.regexp contient deux classes et une exception pour gérer les expressions régulières :

Classe	Rôle
Matcher	comparer une chaîne de caractère avec un motif
Pattern	encapsule une version compilée d'un motif
PatternSyntaxException	exception levée lorsque le motif contient une erreur de syntaxe

5.5.7.1. Les motifs

Les expressions régulières utilisent un motif. Ce motif est une chaîne de caractères qui contient des caractères et des métacaractères. Les métacaractères ont une signification particulière et sont interprétés.

Il est possible de déspécialiser un métacaractère (lui enlever sa signification particulière) en le faisant précéder d'un caractère backslash. Ainsi pour utiliser le caractère backslash, il faut le doubler.

Les métacaractères reconnus par l'api sont :

métacaractères	rôle
()	
[]	définir un ensemble de caractères
{}	définir une répétition du motif précédent
\	déspécalisation du caractère qui suit
^	début de la ligne
$	fin de la ligne
|	le motif précédent ou le motif suivant
?	motif précédent répété zero ou une fois
*	motif précédent repété zéro ou plusieurs fois
+	motif précédent répété une ou plusieurs fois
.	un caractère quelconque

Certains caractères spéciaux ont une notation particulière :

Notation	Rôle
\t	tabulation
\n	nouvelle ligne (ligne feed)
\\	backslash

Il est possible de définir des ensembles de caractères à l'aide des caractères [et]. Il suffit d'indiquer les caractères de l'ensemble entre ces deux caractères

Exemple : toutes les voyelles
`[aeiouy]`

Il est possible d'utiliser une plage de caractères consécutifs en séparant le caractère de début de la plage et le caractère de fin de la plage avec un caractère −

Exemple : toutes les lettres minuscules
`[a-z]`

L'ensemble peut être l'union de plusieurs plages.

Exemple : toutes les lettres
`[a-zA-Z]`

Par défaut l'ensemble [] désigne tous les caractères. Il est possible de définir un ensemble de la forme tous sauf ceux précisés en utilisant le caractère ^ suivi des caractères à enlever de l'ensemble

Exemple : tous les caractères sauf les lettres
`[^a-zA-Z]`

Il existe plusieurs ensembles de caractères prédéfinis :

Notation	Contenu de l'ensemble
\d	un chiffre
\D	tous sauf un chiffre
\w	une lettre ou un underscore
\W	tous sauf une lettre ou un underscore
\s	un séparateur (espace, tabulation, retour chariot, ...)
\S	tous sauf un séparateur

Plusieurs métacaractères permettent de préciser un critère de répétition d'un motif

métacaractères	rôle
{n}	répétition du motif précédent n fois
{n,m}	répétition du motif précédent entre n et m fois

{n,}	répétition du motif précédent
?	motif precédent répété zero ou une fois
*	motif précédent repeté zéro ou plusieurs fois
+	motif précédent répété une ou plusieurs fois

Exemple : la chaîne AAAAA

```
A{5}
```

5.5.7.2. La classe Pattern

Cette classe encapsule une representation compilée d'un motif d'une expression régulière.

La classe Pattern ne possède pas de constructeur public mais propose une méthode statique compile().

Exemple :

```
	private static Pattern motif = null;
	...
	motif = Pattern.compile("liste[0-9]");
```

Une version surchargée de la méthode compile() permet de préciser certaines options dont la plus interressante permet de rendre insensible à la casse les traitements en utilisant le flag CASE_INSENSITIVE.

Exemple :

```
	private static Pattern motif = null;
	...
	motif = Pattern.compile("liste[0-9]",Pattern.CASE_INSENSITIVE);
```

Cette méthode compile() renvoie une instance de la classe Pattern si le motif est syntaxiquement correcte sinon elle lève une exception de type PatternSyntaxException.

La méthode matches(String, String) permet de rapidement et facilement utiliser les expressions régulières avec un seul appel de méthode en fournissant le motif est la chaîne à traiter.

Exemple :

```
	if (Pattern.matches("liste[0-9]","liste2")) {
		System.out.println("liste2 ok");
	} else {
		System.out.println("liste2 ko");
	}
```

5.5.7.3. La classe Matcher

La classe Matcher est utilisée pour effectuer la comparaison entre une chaîne de caractères et un motif encapsulé dans un objet de type Pattern.

Cette classe ne possède aucun constructeur public. Pour obtenir une instance de cette classe, il faut utiliser la méthode matcher() d'une instance d'un objet Pattern en lui fournissant la chaîne à traiter en paramètre.

Exemple :

```
motif = Pattern.compile("liste[0-9]");
matcher = motif.matcher("liste1");
```

La méthodes matches() tente de comparer toute la chaîne avec le motif et renvoie le résultat de cette comparaison.

Exemple :

```
motif = Pattern.compile("liste[0-9]");
matcher = motif.matcher("liste1");
if (matcher.matches()) {
      System.out.println("liste1 ok");
} else {
      System.out.println("liste1 ko");
}
matcher = motif.matcher("liste10");
if (matcher.matches()) {
      System.out.println("liste10 ok");
} else {
      System.out.println("liste10 ko");
}
```

Résultat :

```
liste1 ok
liste10 ko
```

La méthode lookingAt() tente de recherche le motif dans la chaîne à traiter

Exemple :

```
motif = Pattern.compile("liste[0-9]");
matcher = motif.matcher("liste1");
if (matcher.lookingAt()) {
      System.out.println("liste1 ok");
} else {
      System.out.println("liste1 ko");
}
matcher = motif.matcher("liste10");
if (matcher.lookingAt()) {
      System.out.println("liste10 ok");
} else {
      System.out.println("liste10 ko");
}
```

Résultat :

```
liste1 ok
 liste10 ok
```

La méthode find() permet d'obtenir des informations sur chaque occurrence ou le motif est trouvé dans la chaîne à traiter.

Exemple :

```
matcher = motif.matcher("zzliste1zz");
if (matcher.find()) {
      System.out.println("zzliste1zz ok");
} else {
     System.out.println("zzliste1zz ko");
}
```

Résultat :

```
zzliste1zz ok
```

Il est possible d'appeler successivement cette méthode pour obtenir chacune des occurrences.

Exemple :

```
int i = 0;
 motif = Pattern.compile("liste[0-9]");
 matcher = motif.matcher("liste1liste2liste3");
 while (matcher.find()) {
       i++;
 }
 System.out.println("nb occurences = " + i);
```

Les méthodes start() et end() permettent de connaitre la position de début et de fin dans la chaîne dans l'occurrence en cours de traitement.

Exemple :

```
int i = 0;
 motif = Pattern.compile("liste[0-9]");
 matcher = motif.matcher("liste1liste2liste3");
 while (matcher.find()) {
       i++;
 }
 System.out.println("nb occurences = " + i);
```

Résultat :

```
pos debut : 0 pos fin : 6
 pos debut : 6 pos fin : 12
 pos debut : 12 pos fin : 18
 nb occurences = 3
```

La classe Matcher propose aussi les méthodes replaceFirst() et replaceAll() pour facilement remplacer la première ou toutes les occurrences du motif trouvé par une chaîne de caractères.

Exemple : remplacement de la première occurrence

```
motif = Pattern.compile("liste[0-9]");
 matcher = motif.matcher("zz liste1 zz liste2 zz");
 System.out.println(matcher.replaceFirst("chaine"));
```

Résultat :

```
zz chaine zz liste2 zz
```

Exemple : remplacement de toutes les occurrences

```
motif = Pattern.compile("liste[0-9]");
matcher = motif.matcher("zz liste1 zz liste2 zz");
System.out.println(matcher.replaceAll("chaine"));
```

Résultat :

```
zz chaine zz chaine zz
```

5.5.8. La classe Formatter

La méthode printf() utilise la classe Formatter pour réaliser le formatage des données fournies selon leurs valeurs et le format donné en paramètre.

Cette classe peut aussi être utilisée pour formatter de données pour des fichiers ou dans une servlet par exemple.

La méthode format() attend en paramètre une chaîne de caractères qui précise le format les données à formater.

Exemple (java 1.5) :

```
import java.util.*;

public class TestFormatter {

  public static void main(String[] args) {
    Formatter formatter = new Formatter();
    formatter.format("%04d \n",13);
    String resultat = formatter.toString();
    System.out.println("chaine = " + resultat);

  }

}
```

Résultat :

```
C:\tiger>java TestFormatter
chaine = 0013
```

5.5.9. La classe Scanner

Cette classe facilite la lecture dans un flux. Elle est particulièrement utile pour réaliser une lecture de données à partir du clavier dans une application de type console.

La méthode next() bloque l'exécution jusqu'à la lecture de données et les renvoie sous la forme d'une chaîne de caractères.

Exemple (java 1.5) :

```
import java.util.*;

public class TestScanner {

  public static void main(String[] args) {
    Scanner scanner = Scanner.create(System.in);
    String chaine = scanner.next();
    scanner.close();

  }

}
```

Cette classe possède plusieurs méthodes nextXXX() ou XXX représente un type primitif. Ces méthodes bloquent l'exécution jusqu'à la lecture de données et tente de les convertir dans le type XXX

Exemple (java 1.5) :

```
import java.util.*;

public class TestScanner {
```

```
  public static void main(String[] args) {
    Scanner scanner = Scanner.create(System.in);
    int entier = scanner.nextInt();
    scanner.close();
  }

}
```

Une exception de type InputMismatchException est levée si les données lue dans le flux

Exemple (java 1.5) :

```
C:\tiger>java TestScanner
texte
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at TestScanner.main(TestScanner.java:8)
```

La classe Scanner peut être utilisée avec n'importe quel flux.

5.6. Présentation rapide du package java.net

Ce package contient un ensemble de classes pour permettre une interaction avec le réseau pour permettre de recevoir et d'envoyer des données à travers ce dernier.

Le chapitre "l'interaction avec le reseau" détaille l'utilisation de ce package.

5.7. Présentation rapide du package java.applet

Ce package contient les classes necéssaires au développement des applets. Une applet est une petite application téléchargée par le réseau et exécutée sous de fortes contraintes de sécurité dans une page Web par le navigateur.

Le développement des applets est détaillé dans le chapitre "les applets"

6. Les fonctions mathématiques

La classe java.lang.Math contient une série de méthodes et variables mathématiques. Comme la classe Math fait partie du package java.lang, elle est automatiquement importée. de plus, il n'est pas nécessaire de déclarer un objet de type Math car les méthodes sont toutes static

Exemple (code Java 1.1) : Calculer et afficher la racine carrée de 3

```
public class Math1 {
  public static void main(java.lang.String[] args) {
    System.out.println(" = " + Math.sqrt(3.0));
  }
}
```

Ce chapitre contient plusieurs sections :

6.1. Les variables de classe

PI représente pi dans le type double (3,14159265358979323846)

E représente e dans le type double (2,7182818284590452354)

Exemple (code Java 1.1) :

```
public class Math2 {
  public static void main(java.lang.String[] args) {
    System.out.println(" PI = "+Math.PI);
    System.out.println(" E = "+Math.E);
  }
}
```

6.2. Les fonctions trigonométriques

Les méthodes sin(), cos(), tan(), asin(), acos(), atan() sont déclarées : public static double fonctiontrigo(double angle)

Les angles doivent être exprimés en radians. Pour convertir des degrés en radian, il suffit de les multiplier par PI/180

6.3. Les fonctions de comparaisons

max (n1, n2)
min (n1, n2)

Ces méthodes existent pour les types int, long, float et double : elles déterminent respectivement les valeurs maximales et minimales des deux paramètres.

Exemple (code Java 1.1) :

```
public class Math1 {
        public static void main(String[] args) {
                System.out.println(" le plus grand = " + Math.max(5, 10));
                System.out.println(" le plus petit = " + Math.min(7, 14));
        }
}
```

Résultat :

```
le plus grand = 10
le plus petit = 7
```

6.4. Les arrondis

La classe Math propose plusieurs méthodes pour réaliser différents arrondis.

6.4.1. La méthode round(n)

Cette méthode ajoute 0,5 à l'argument et restitue la plus grande valeur entière (int) inférieure ou égale au résultat. La méthode est définie pour les types float et double.

Exemple (code Java 1.1) :

```
public class Arrondis1 {
  static double[] valeur = {-5.7, -5.5, -5.2, -5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i <valeur.length; i++) {
      System.out.println("round("+valeur[i]+") = "+Math.round(valeur[i]));
    }
  }
}
```

Résultat :

```
round(-5.7) = -6
round(-5.5) = -5
round(-5.2) = -5
round(-5.0) = -5
round(5.0) = 5
round(5.2) = 5
round(5.5) = 6
round(5.7) = 6
```

6.4.2. La méthode rint(double)

Cette méthode effectue la même opération mais renvoie un type double.

Exemple (code Java 1.1) :

```
public class Arrondis2 {
  static double[] valeur = {−5.7, −5.5, −5.2, −5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i <valeur.length; i++) {
    System.out.println("rint("+valeur[i]+") = "+Math.rint(valeur[i]));
    }
  }
}
```

Résultat :

```
rint(−5.7) = −6.0
rint(−5.5) = −6.0
rint(−5.2) = −5.0
rint(−5.0) = −5.0
rint(5.0) = 5.0
rint(5.2) = 5.0
rint(5.5) = 6.0
rint(5.7) = 6.0
```

6.4.3. La méthode floor(double)

Cette méthode renvoie l'entier le plus proche inférieur ou égal à l'argument

Exemple (code Java 1.1) :

```
public class Arrondis3 {
  static double[] valeur = {−5.7, −5.5, −5.2, −5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i <valeur.length; i++) {
      System.out.println("floor("+valeur[i]+") = "+Math.floor(valeur[i]));
    }
  }
}
```

Résultat :

```
floor(−5.7) = −6.0
floor(−5.5) = −6.0
floor(−5.2) = −6.0
floor(−5.0) = −5.0
floor(5.0) = 5.0
floor(5.2) = 5.0
floor(5.5) = 5.0
floor(5.7) = 5.0
```

6.4.4. La méthode ceil(double)

Cette méthode renvoie l'entier le plus proche supérieur ou égal à l'argument

Exemple (code Java 1.1) :

```
public class Arrondis4 {
  static double[] valeur = {−5.7, −5.5, −5.2, −5.0, 5.0, 5.2, 5.5, 5.7 };
```

```
  public static void main(String[] args) {
    for (int i = 0; i <valeur.length; i++) {
      System.out.println("ceil("+valeur[i]+") = "+Math.ceil(valeur[i]));
    }
  }
}
```

résultat :

```
ceil(−5.7) = −5.0
ceil(−5.5) = −5.0
ceil(−5.2) = −5.0
ceil(−5.0) = −5.0
ceil(5.0) = 5.0
ceil(5.2) = 6.0
ceil(5.5) = 6.0
ceil(5.7) = 6.0
```

6.4.5. La méthode abs(x)

Cette méthode donne la valeur absolue de x (les nombre négatifs sont convertis en leur opposé). La méthode est définie pour les types int, long, float et double.

Exemple (code Java 1.1) :

```
public class Math1 {
  public static void main(String[] args) {
    System.out.println(" abs(−5.7) = "+abs(−5.7));
  }
}
```

Résultat :

```
abs(−5.7) = 5.7
```

6.5. La méthode IEEEremainder(double, double)

Cette méthode renvoie le reste de la division du premier argument par le deuxieme

Exemple (code Java 1.1) :

```
public class Math1 {
        public static void main(String[] args) {
        System.out.println(" reste de la division de 3 par 10 = "
           +Math.IEEEremainder(10.0, 3.0) );
        }
}
```

Résultat :

```
reste de la division de 3 par 10 = 1.0
```

6.6. Les Exponentielles et puissances

6.6.1. La méthode pow(double, double)

Cette méthode elève le premier argument à la puissance indiquée par le second.

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
        System.out.println(" 5 au cube  = "+Math.pow(5.0, 3.0) );
}
```

Résultat :

```
5 au cube  = 125.0
```

6.6.2. La méthode sqrt(double)

Cette méthode calcule la racine carrée de son paramètre.

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
  System.out.println(" racine carree de 25  = "+Math.sqrt(25.0) );
}
```

Résultat :

```
racine carree de 25  = 5.0
```

6.6.3. La méthode exp(double)

Cette méthode calcule l'exponentielle de l'argument

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
        System.out.println(" exponentiel de 5  = "+Math.exp(5.0) );
}
```

Résultat :

```
exponentiel de 5  = 148.4131591025766
```

6.6.4. La méthode log(double)

Cette méthode calcule le logarithme naturel de l'argument

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
        System.out.println(" logarithme de 5  = "+Math.log(5.0) );
}
```

Résultat :

```
logarithme de 5  = 1.6094379124341003
```

6.7. La génération de nombres aléatoires

6.7.1. La méthode random()

Cette méthode renvoie un nombre aléatoire compris entre 0.0 et 1.0.

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
        System.out.println(" un nombre aléatoire  = "+Math.random() );
}
```

Résultat :

```
un nombre aléatoire  = 0.8178819778125899
```

7. La gestion des exceptions

Les exceptions représentent le mécanisme de gestion des erreurs intégré au langage java. Il se compose d'objets représentant les erreurs et d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs (try, catch et finally) et de les lever ou les propager (throw et throws).

Lors de la détection d'une erreur, un objet qui hérite de la classe Exception est créé (on dit qu'une exception est levée) et propagé à travers la pile d'execution jusqu'à ce qu'il soit traité.

Ces mécanismes permettent de renforcer la sécurité du code Java.

Exemple (code Java 1.1) : une exception levée à l'exécution non capturée

```
public class TestException {
   public static void main(java.lang.String[] args) {
      int i = 3;
      int j = 0;
      System.out.println("résultat = " + (i / j));
   }
}
```

Résultat :

```
C:>java TestException
Exception in thread "main" java.lang.ArithmeticException: /
by zero
        at tests.TestException.main(TestException.java:23)
```

Si dans un bloc de code on fait appel à une méthode qui peut potentiellement générer une exception, on doit soit essayer de la récupérer avec try/catch, soit ajouter le mot clé throws dans la déclaration du bloc. Si on ne le fait pas, il y a une erreur à la compilation. Les erreurs et exceptions du paquetage java.lang échappent à cette contrainte. Throws permet de déléguer la responsabilité des erreurs vers la méthode appelante

Ce procédé présente un inconvénient : de nombreuses méthodes des packages java indiquent dans leur déclaration qu'elles peuvent lever une exception. Cependant ceci garantie que certaines exceptions critiques seront prises explicitement en compte par le programmeur.

Ce chapitre contient plusieurs sections :

7.1. Les mots clés try, catch et finally

Le bloc try rassemble les appels de méthodes susceptibles de produire des erreurs ou des exceptions. L'instruction try est suivie d'instructions entre des accolades.

Exemple (code Java 1.1) :

```
   try {
        operation_risquée1;
        opération_risquée2;
    } catch (ExceptionInteressante e) {
        traitements
    } catch (ExceptionParticulière e) {
        traitements
    } catch (Exception e) {
        traitements
    } finally {
        traitement_pour_terminer_proprement;
    }
```

Si un événement indésirable survient dans le bloc try, la partie éventuellement non exécutée de ce bloc est abandonnée et le premier bloc catch est traité. Si catch est défini pour capturer l'exception issue du bloc try alors elle est traitée en exécutant le code associé au bloc. Si le bloc catch est vide (aucune instruction entre les accolades) alors l'exception capturée est ignorée. Une telle utilisation de l'instruction try/catch n'est pas une bonne pratique : il est préférable de toujours apporter un traitement adapté lors de la capture d'une exception.

Si il y a plusieurs type d'erreurs et d'exceptions à intercepter, il faut définir autant de bloc catch que de type d'événement . Par type d'exception, il faut comprendre « qui est du type de la classe de l'exception ou d'une de ces sous classes ». Ainsi dans l'ordre séquentiel des clauses catch, un type d'exception de ne doit pas venir après un type d'une exception d'une super classe. Il faut faire attention à l'ordre des clauses catch pour traiter en premier les exceptions les plus précises (sous classes) avant les exceptions plus générales. Un message d'erreur est émis par le compilateur dans le cas contraire.

Exemple (code Java 1.1) : erreur à la compil car Exception est traité en premier alors que ArithmeticException est une sous classe de Exception

```
public class TestException {
    public static void main(java.lang.String[] args) {
        // Insert code to start the application here.
        int i = 3;
        int j = 0;
        try {
            System.out.println("résultat = " + (i / j));
        }
        catch (Exception e) {
        }
        catch (ArithmeticException e) {
        }
    }
}
```

Résultat :

```
C:\tests>javac TestException.java
TestException.java:11: catch not reached.
        catch (ArithmeticException e) {
        ^
1 error
```

Si l'exception générée est une instance de la classe déclarée dans la clause catch ou d'une classe dérivée, alors on exécute le bloc associé. Si l'exception n'est pas traitées par un bloc catch, elle sera transmise au bloc de niveau supérieur. Si l'on ne se trouve pas dans un autre bloc try, on quitte la méthode en cours, qui regénère à son tour une exception dans la

méthode appelante.

L'exécution totale du bloc try et d'un bloc d'une clause catch sont mutuellement exclusives : si une exception est levée, l'exécution du bloc try est arrêtée et si elle existe, la clause catch adéquate est exécutée.

La clause finally définit un bloc qui sera toujours exécuté, qu'une exception soit levée ou non. Ce bloc est facultatif. Il est aussi exécuté si dans le bloc try il y a une instruction break ou continue.

7.2. La classe Throwable

Cette classe descend directement de Object : c'est la classe de base pour le traitements des erreurs.

Cette classe possède deux constructeurs :

Méthode	Rôle
Throwable()	
Throwable(String)	La chaîne en paramètre permet de définir un message qui décrit l'exception et qui pourra être consultée dans un bloc catch.

Les principales méthodes de la classe Throwable sont :

Méthodes	Rôle
String getMessage( )	lecture du message
void printStackTrace( )	affiche l'exception et l'état de la pile d'execution au moment de son appel
void printStackTrace(PrintStream s)	Idem mais envoie le résultat dans un flux

Exemple (code Java 1.1) :

```
public class TestException {
    public static void main(java.lang.String[] args) {
        // Insert code to start the application here.
        int i = 3;
        int j = 0;
        try {
            System.out.println("résultat = " + (i / j));
        }
        catch (ArithmeticException e) {
            System.out.println("getmessage");
            System.out.println(e.getMessage());
            System.out.println(" ");
            System.out.println("toString");
            System.out.println(e.toString());
            System.out.println(" ");
            System.out.println("printStackTrace");
            e.printStackTrace();
        }
    }
}
```

Résultat :

```
C:>java TestException
getmessage
/ by zero

toString
```

```
java.lang.ArithmeticException: / by zero

printStackTrace
java.lang.ArithmeticException: / by zero
        at tests.TestException.main(TestException.java:24)
```

7.3. Les classes Exception, RunTimeException et Error

Ces trois classes descendent de Throwable : en fait, toutes les exceptions dérivent de la classe Throwable.

La classe Error représente une erreur grave intervenue dans la machine virtuelle Java ou dans un sous système Java. L'application Java s'arrête instantanement dès l'apparition d'une exception de la classe Error.

La classe Exception représente des erreurs moins graves. Les exceptions héritant de classe RuntimeException n'ont pas besoin d'être détectées impérativement par des blocs try/catch.

7.4. Les exceptions personnalisées

Pour générer une exception, il suffit d'utiliser le mot clé throw, suivi d'un objet dont la classe dérive de Throwable. Si l'on veut générer une exception dans une méthode avec throw, il faut l'indiquer dans la déclaration de la méthode, en utilisant le mot clé throws.

En cas de nécessité, on peut créer ses propres exceptions. Elles descendent des classes Exception ou RunTimeException mais pas de la classe Error. Il est préférable (par convention) d'inclure le mot « Exception » dans le nom de la nouvelle classe.

Exemple (code Java 1.1) :

```
public class SaisieErroneeException extends Exception {
    public SaisieErroneeException() {
        super();
    }
    public SaisieErroneeException(String s) {
        super(s);
    }
}

public class TestSaisieErroneeException {
    public static void controle(String chaine) throws
SaisieErroneeException {
        if (chaine.equals("") == true)
            throw new SaisieErroneeException("Saisie erronee : chaine vide");
    }
    public static void main(java.lang.String[] args) {
        String chaine1 = "bonjour";
        String chaine2 = "";
        try {
            controle(chaine1);
        }
        catch (SaisieErroneeException e) {
            System.out.println("Chaine1 saisie erronee");
        };
        try {
            controle(chaine2);
        }
        catch (SaisieErroneeException e) {
            System.out.println("Chaine2 saisie erronee");
        };
    }
}
```

Les méthodes pouvant lever des exceptions doivent inclure une clause throws nom_exception dans leur en tête. L'objectif est double : avoir une valeur documentaire et préciser au compilateur que cette méthode pourra lever cette exception et que toute méthode qui l'appelle devra prendre en compte cette exception (traitement ou propagation).

Si la méthode appelante ne traite pas l'erreur ou ne la propage pas, le compilateur génère l'exception nom_exception must be caught or it must be déclared in the thows clause of this méthode.

Java n'oblige la déclaration des exceptions dans l'en tête de la méthode que pour les exceptions dites controlées (checked). Les exceptions non controlées (unchecked) peuvent être capturées mais n'ont pas a être déclarées. Les exceptions et erreurs qui héritent de RunTimeException et de Error sont non controlées. Toutes les autres exceptions sont controlées.

8. Le multitâche

Un thread est une unité d'éxecution faisant partie d'un programme. Cette unité fonctionne de façon autonome et parallèlement à d'autres threads. En fait, sur une machine mono processeur, chaque unité se voit attribuer des intervalles de temps au cours desquels elles ont le droit d'utiliser le processeur pour accomplir leurs traitements.

La gestion de ces unités de temps par le système d'exploitation est appellée scheduling. Il existe deux grands types de scheduller:

- le découpage de temps : utilisé par Windows et Macintosh OS jusqu'à la version 9. Ce système attribue un intervalle de temps prédéfini quelque soit le thread et la priorité qu'il peut avoir
- la préemption : utilisé par les systèmes de type Unix. Ce système attribut les intervalles de temps en tenant compte de la priorité d'exécution de chaque thread. Les threads possédant une priorité plus élevée s'exécutent avant ceux possédant une priorité plus faible.

Le principal avantage des threads est de pouvoir répartir différents traitements d'un même programme en plusieurs unités distinctes pour permettre leur exécution "simultanée".

La classe java.lang.Thread et l'interface java.lang.Runnable sont les bases pour le développement des threads en java. Par exemple, pour exécuter des applets dans un thread, il faut que celles ci implémentent l'interface Runnable.

Le cycle de vie d'un thread est toujours le même qu'il hérite de la classe Thread ou qu'il implémente l'interface Runnable. L'objet correspondant au thread doit être créé, puis la méthode start() est appelée qui à son tour invoque la méthode run(). La méthode stop() permet d'interrompre le thread.

Avant que le thread ne s'exécute, il doit être démarré par un appel à la méthode start(). On peut créer l'objet qui encapsule le thread dans la méthode start() d'une applet, dans sa méthode init() ou dans le constructeur d'une classe.

Ce chapitre contient plusieurs sections :

8.1. L'interface Runnable

Cette interface doit être implémentée par toute classe qui contiendra des traitements à exécuter dans un thread.

Cette interface ne définit qu'une seule méthode : void run().

Dans les classes qui implémentent cette interface, la méthode run() doit être redéfinie pour contenir le code des traitements qui seront exécutés dans le thread.

Exemple :

```
package com.moi.test;

public class MonThread3 implements Runnable {

  public void run() {
    int i = 0;
    for (i = 0; i <10; i++) {
      System.out.println("" + i);
    }
  }
}
```

Lors du démarrage du thread, la méthode run() est appelée.

8.2. La classe Thread

La classe Thread est définie dans le package java.lang. Elle implémente l'interface Runnable.

Elle possède plusieurs constructeurs : un constructeur par défaut et plusieurs autres qui peuvent avoir un ou plusieurs des paramètres suivants :

Paramètre	Rôle
un nom	le nom du thread : si aucun n'est précisé alors le nom sera thread−nnn ou nnn est un numéro séquentiel
un objet qui implémente l'interface Runnable	l'objet qui contient les traitements du thread
un groupe	le groupe auquel sera rattaché le thread

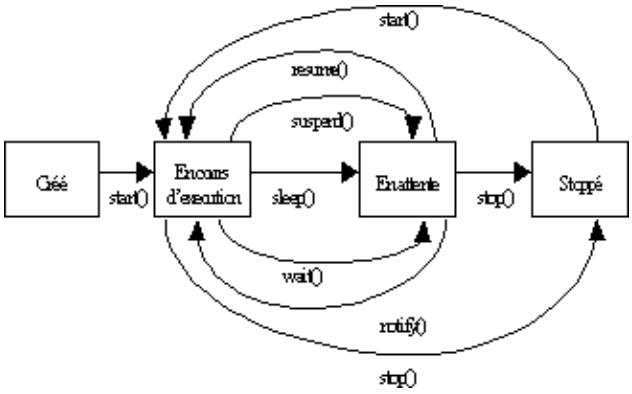
Un thread possède une priorité et un nom. Si aucun nom particulier n'est donné dans le constructeur du thread, un nom par défaut composé du suffixe "Thread−" suivi d'un numéro séquentiel incrémenté automatiquement lui est attribué.

La classe thread possède plusieurs méthodes pour gérer le cycle de vie du thread.

Méthode	Rôle
void destroy()	met fin brutalement au thread : à n'utiliser qu'en dernier recours.
int getPriority()	renvoie la priorité du thread
ThreadGroup getThreadGroup()	renvoie un objet qui encapsule le groupe auquel appartient le thread
boolean isAlive()	renvoie un booléen qui indique si le thread est actif ou non
boolean isInterrupted()	renvoie un booléen qui indique si le thread a été interrompu
void join()	
void resume()	reprend l'exécution du thread( ) préalablement suspendu par suspend( ). Cette méthode est dépréciée
void run()	méthode déclarée par l'interface Runnable : elle doit contenir le code qui sera exécuté par le thread
void sleep(long)	mettre le thread en attente durant le temps exprimé en millisecondes fourni en paramètre. Cette méthode peut lever une exception de type InterruptedException si le thread est réactivé avant la fin du temps.
void start()	démarrer le thread et exécuter la méthode run()

void stop()	arrêter le thread. Cette méthode est dépréciée
void suspend()	suspend le thread jusqu'au moment où il sera relancé par la méthode resume( ). Cette méthode est dépréciée
void yield()	indique à l'interpréteur que le thread peut être suspendu pour permettre à d'autres threads de s'exécuter.

Le cycle de vie avec le JDK 1.0 est le suivant :



Le comportement de la méthode start() de la classe Thread dépend de la façon dont l'objet est instancié. Si l'objet qui reçoit le message start() est instancié avec un constructeur qui prend en paramètre un objet Runnable, c'est la méthode run() de cet objet qui est appelée. Si l'objet qui reçoit le message start() est instancié avec un constructeur qui ne prend pas en paramètre une référence sur un objet Runnable, c'est la méthode run() de l'objet qui reçoit le message start() qui est appelée.

A partir du J.D.K. 1.2, les méthodes stop(), suspend() et resume() sont dépréciées. Le plus simple et le plus efficace est de définir un attribut booléen dans la classe du thread initialisé à true. Il faut définir une méthode qui permet de basculer cet attribut à false. Enfin dans la méthode run() du thread, il suffit de continuer les traitements tant que l'attribut est à true et que les autres conditions fonctionnelles d'arrêt du thread sont négatives.

Exemple : exécution du thread jusqu'à l'appui sur la touche Entrée

```
public class MonThread6 extends Thread {
  private boolean actif = true;

  public static void main(String[] args) {
    try {
      MonThread6 t = new MonThread6();
      t.start();
      System.in.read();
      t.arreter();
    } catch (Exception e) {
      e.printStackTrace();
    }

  }

  public void run() {
    int i = 0;
    while (actif) {
      System.out.println("i = " + i);
      i++;
    }
  }

  public void arreter() {
    actif = false;
  }
}
```

Si la méthode start() est appelée alors que le thread est déjà en cours d'exécution, une exception de type IllegalThreadStateException est levée.

Exemple :

```
package com.moi.test;

public class MonThread5 {

  public static void main(String[] args) {
    Thread t = new Thread(new MonThread3());
    t.start();
    t.start();
  }
}
```

Résultat :

```
java.lang.IllegalThreadStateException
      at java.lang.Thread.start(Native Method)
      at com.moi.test.MonThread5.main(MonThread5.java:14)
Exception in thread "main"
```

La méthode sleep() permet d'endormir le thread durant le temps en millisecondes fournis en paramètres de la méthode.

La méthode statique currentThread() renvoie le thread en cours d'exécution.

La méthode isAlive() renvoie un booléen qui indique si le thread est en cours d'exécution.

8.3. La création et l'exécution d'un thread

Pour que les traitements d'une classe soient exécutés dans un thread, il faut obligatoirement que cette classe implémente l'interface Runnable puis que celle ci soit associée directement ou indirectement à un objet de type Thread

Il y a ainsi deux façons de définir une telle classe

- la classe hérite de la classe Thread
- la classe implémente l'interface Runnable

8.3.1. La dérivation de la classe Thread

Le plus simple pour définir un thread est de créer une classe qui hérite de la classe java.lang.Thread.

Il suffit alors simplement de redéfinir la méthode run() pour y inclure les traitements à exécuter par le thread.

Exemple :

```
package com.moi.test;

public class MonThread2 extends Thread {

  public void run() {
    int i = 0;
    for (i = 0; i <10; i++) {
      System.out.println("" + i);
    }
  }
```

```
}
```

Pour créer et exécuter un tel thread, il faut instancier un objet et appeler sa méthode start(). Il est obligatoire d'appeler la méthode start() qui va créer le thread et elle−même appeler la méthode run().

Exemple :

```
package com.moi.test;

public class MonThread2 extends Thread {

  public static void main(String[] args) {
        Thread t = new MonThread2();
        t.start();
  }

  public void run() {
    int i = 0;
    for (i = 0; i <10; i++) {
      System.out.println("" + i);
    }
  }

}
```

8.3.2. Implémentation de l'interface Runnable

Si on utilise l'interface Runnable , il faut uniquement redéfinir sa seule et unique méthode run() pour y inclure les traitements à exécuter dans le thread.

Exemple :

```
package com.moi.test;

public class MonThread3 implements Runnable {

  public void run() {
    int i = 0;
    for (i = 0; i <10; i++) {
      System.out.println("" + i);
    }
  }
}
```

Pour pouvoir utiliser cette classe dans un thread, il faut l'associer à un objet de la classe Thread. Ceci ce fait en utilisant un des constructeurs de la classe Thread qui accepte un objet implémentant l'interface Runnable en paramètre.

Exemple :

```
package com.moi.test;

public class LancerDeMonThread3 {

  public static void main(String[] args) {
    Thread t = new Thread(new MonThread3());
    t.start();
  }
}
```

Il ne reste plus alors qu'à appeller la méthode start() du nouvel objet.

8.3.3. Modification de la priorité d'un thread

Lors de la création d'un thread, la priorité du nouveau thread est égale à celle du thread dans lequel il est créé. Si le thread n'est pas créé dans un autre thread, la priorité moyenne est attribué au thread. Il est cependant possible d'attribuer une autre priorité plus ou moins élevée.

En java, la gestion des threads est intimement liée au système d'exploitation dans lequel s'exécute la machine virtuelle. Sur des machines de type Mac ou Unix, le thread qui a la plus grande priorité a systématiquement accès au processeur si il ne se trouve pas en mode « en attente ». Sous Windows 95, le système ne gère pas correctement les priorités et il choisit lui même le thread a exécuter : l'attribution d'un priorité supérieure permet simplement d'augmenter ses chances d'exécution.

La priorité d'un thread varie de 1 à 10 , la valeur 5 étant la valeur par défaut. La classe Thread définit trois constantes :

MIN_PRIORITY : priorité inférieure

NORM_PRIORITY : priorité standard

MAX_PRIORITY : priorité supérieure

Exemple :

```
package com.moi.test;

public class TestThread10 {

  public static void main(String[] args) {
    System.out.println("Thread.MIN_PRIORITY  = " + Thread.MIN_PRIORITY);
    System.out.println("Thread.NORM_PRIORITY = " + Thread.NORM_PRIORITY);
    System.out.println("Thread.MAX_PRIORITY  = " + Thread.MAX_PRIORITY);
  }
}
```

Résultat :

```
Thread.MIN_PRIORITY  = 1
Thread.NORM_PRIORITY = 5
Thread.MAX_PRIORITY  = 10
```

Pour déterminer ou modifier la priorité d'un thread, la classe Thread contient les méthodes suivantes :

Méthode	Rôle
int getPriority()	retourne la priorité d'un thread
void setPriority(int)	modifie la priorité d'un thread

La méthode setPriority() peut lever l'exception IllegalArgumentException si la priorité fournie en paramètre n'est pas comprise en 1 et 10.

Exemple :

```
package com.moi.test;

public class TestThread9 {

  public static void main(String[] args) {
    Thread t = new Thread();

    t.setPriority(20);
  }
```

```
}
```

Résultat :

```
java.lang.IllegalArgumentException
        at java.lang.Thread.setPriority(Unknown Source)
        at com.moi.test.MonThread9.main(TestThread9.java:8)
Exception in thread "main"
```

8.4. La classe ThreadGroup

La classe ThreadGroup représente un ensemble de threads. Il est ainsi possible de regrouper des threads selon différents critères. Il suffit de créer un objet de la classe ThreadGroup et de lui affecter les différents threads. Un objet ThreadGroup peut contenir des threads mais aussi d'autres objets de type ThreadGroup.

La notion de groupe permet de limiter l'accès aux autres threads. Chaque thread ne peut manipuler que les threads de son groupe d'appartenance ou des groupes subordonnés.

La classe ThreadGroup possède deux constructeurs :

Constructeur	Rôle
ThreadGroup(String nom)	création d'un groupe avec attribution d'un nom
ThreadGroup(ThreadGoup groupe_parent, String nom)	création d'un groupe à l'intérieur du groupe spécifié avec l'attribution d'un nom

Pour ajouter un thread à un groupe, il suffit de préciser le groupe en paramètre du constructeur du thread.

Exemple :

```
package com.moi.test;

public class LanceurDeThreads {

  public static void main(String[] args) {
    ThreadGroup tg = new ThreadGroup("groupe");
    Thread t1 = new Thread(tg,new MonThread3(), "numero 1");
    Thread t2 = new Thread(tg,new MonThread3(), "numero 2");
  }
}
```

L'un des avantages de la classe ThreadGroup est de permettre d'effectuer une action sur tous les threads d'un même groupe. On peut, par exemple avec Java 1.0, arrêter tous les threads du groupe en lui appliquant la méthode stop().

8.5. Thread en tâche de fond (démon)

Il existe une catégorie de threads qualifiés de démons : leur exécution peut se poursuivre même après l'arrêt de l'application qui les a lancés.

Une application dans laquelle les seuls threads actifs sont des démons est automatiquement fermée.

Le thread doit d'abord être créé comme thread standard puis transformé en demon par un appel à la méthode setDaemon() avec le paramètre true. Cet appel se fait avant le lancement du thread, sinon une exception de type IllegalThreadStateException est levée.

8.6. Exclusion mutuelle

Chaque fois que plusieurs threads s'exécutent en même temps, il faut prendre des précautions concernant leur bonne exécution. Par exemple, si deux threads veulent accéder à la même variable, il ne faut pas qu'ils le fassent en même temps.

Java offre un système simple et efficace pour réaliser cette tache. Si une méthode déclarée avec le mot clé synchronized est déjà en cours d'exécution, alors les threads qui en auraient également besoin doivent attendre leur tour.

Le mécanisme d'exclusion mutuelle en Java est basé sur le moniteur. Pour définir une méthode protégée, afin de s'assurer de la cohérence des données, il faut utiliser le mot clé synchronized. Cela crée à l'exécution, un moniteur associé à l'objet qui empêche les méthodes déclarées synchronized d'être utilisées par d'autres objets dès lors qu'un objet utilise déjà une des méthodes synchronisées de cet objet. Dès l'appel d'une méthode synchronisée, le moniteur vérouille tous les autres appels de méthodes synchronisées de l'objet. L'accès est de nouveau automatiquement possible dès la fin de l'exécution de la méthode.

Ce procédé peut bien évidemment dégrader les performances lors de l'exécution mais il garantit, dès lors qu'il est correctement utilisé, la cohérence des données.

8.6.1. Sécurisation d'une méthode

Lorsque l'on crée une instance d'une classe, on crée également un moniteur qui lui est associé. Le modificateur synchronized place la méthode (le bloc de code) dans ce moniteur, ce qui assure l'exclusion mutuelle

Le méthode ainsi déclarée ne peut être exécutée par plusieurs processus simultanement. Si le moniteur est occupé, les autres processus seront mis en attente. L'ordre de réveil des processus pour accéder à la méthode n'est pas prévisible.

Si un objet dispose de plusieurs méthodes synchronized, ces dernières ne peuvent être appelées que par le thread possédant le verrou sur l'objet.

8.6.2. Sécurisation d'un bloc

L'utilisation de méthodes synchronisées trop longues à exécuter peut entrainer une baisse d'efficacité lors de l'exécution. Avec java, il est possible de placer n'importe quel bloc de code dans un moniteur pour permettre de réduire la longueur des sections de code sensibles.

```
synchronized  void methode1() {
   // bloc de code sensible
   ...
}

      void methode2(Object obj) {
   ...
   synchronized (obj) {
      // bloc de code sensible
      ...
   }
}
```

L'objet dont le moniteur est à utiliser doit être passé en paramètre de l'instruction synchronized .

8.6.3. Sécurisation de variables de classes

Pour sécuriser une variable de classe, il faut un moniteur commun à toutes les instances de la classe. La méthode getClass() retourne la classe de l'instance dans laquelle on l'appelle. Il suffit d'utiliser un moniteur qui utilise le résultat de getClass() comme verrou.

8.6.4. La synchronisation : les méthodes wait() et notify()

La suite de ce chapitre sera développée dans une version future de ce document

9. JDK 1.5 (nom de code Tiger)

La version 1.5 de Java dont le nom de code est Tiger est développée par la JSR 176.

La version utilisée dans ce chapitre est la version bêta 1.

Exemple :

```
C:\>java −version
java version "1.5.0−beta"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0−beta−b32c)
Java HotSpot(TM) Client VM (build 1.5.0−beta−b32c, mixed mode)
```

La version 1.5 de Java apporte de nombreuses évolutions qui peuvent être classées dans deux catégories :

- Les évolutions sur la syntaxe du langage
- Les évolutions sur les API : mises à jour d'API existantes, intégration d'API dans le SDK

Ce chapitre va détailler les nombreuses évolutions sur la syntaxe du langage.

9.1. Les nouveautés du langage Java version 1.5

Depuis sa première version et jusqu'à sa version 1.5, le langage Java lui−même n'a que très peu évolué : la version 1.1 a ajouté les classes internes et la version 1.4 les assertions.

Les évolutions de ces différentes versions concernaient donc essentiellement les API de la bibliothèque standard (core) de Java.

La version 1.5 peut être considérée comme une petite révolution pour Java car elle apporte énormément d'améliorations sur le langage. Toutes ces évolutions sont déjà présentes dans différents autres langages notamment C#.

Le but principal de ces ajouts est de faciliter le développement d'applications avec Java en simplifiant l'écriture et la lecture du code.

Un code utilisant les nouvelles fonctionnalités de Java 1.5 ne pourra pas être exécuté dans une version antérieure de la JVM.

Pour compiler des classes utilisant les nouvelles fonctionnalités de la version 1.5, il faut utiliser les options −target 1.5 et −source 1.5 de l'outil javac. Par défaut, ce compilateur utilise les spécifications 1.4 de la plate−forme.

9.2. Autoboxing / unboxing

L'autoboxing permet de transformer automatiquement une variable de type primitif en un objet du type du wrapper correspondant. L'unboxing est l'opération inverse. Cette nouvelle fonctionnalité est spécifiée dans la JSR 201.

Par exemple, jusqu'à la version 1.4 de Java pour ajouter des entiers dans une collection, il est nécessaire d'encapsuler chaque valeur dans un objet de type Integer.

Exemple :

```
import java.util.*;

public class TestAutoboxingOld {

  public static void main(String[] args) {

    List liste = new ArrayList();
    Integer valeur = null;
    for(int i = 0; i < 10; i++) {
      valeur = new Integer(i);
      liste.add(valeur);
    }

  }

}
```

Avec la version 1.5, l'encapsulation de la valeur dans un objet n'est plus obligatoire car elle sera réalisée automatiquement par le compilateur.

Exemple (java 1.5):

```
import java.util.*;

public class TestAutoboxing {

  public static void main(String[] args) {
    List liste = new ArrayList();
    for(int i = 0; i < 10; i++) {
      liste.add(i);
    }
  }

}
```

9.3. Static import

Jusqu'à la version 1.4 de Java, pour utiliser un membre statique d'une classe, il faut obligatoirement préfixer ce membre par le nom de la classe qui le contient.

Par exemple, pour utiliser la constante Pi définie dans la classe java.lang.Math, il est nécessaire d'utiliser Math.PI

Exemple :

```
public class TestStaticImportOld {

  public static void main(String[] args) {
    System.out.println(Math.PI);
    System.out.println(Math.sin(0));
  }

}
```

Java 1.5 propose une solution pour réduire le code à écrire concernant les membres statiques en proposant une nouvelle fonctionnalité concernant l'importation de package : l'import statique (static import).

Ce nouveau concept permet d'appliquer les mêmes règles aux membres statiques qu'aux classes et interfaces pour l'importation classique.

Cette nouvelle fonctionnalité est développée dans la JSR 201. Elle s'utilise comme une importation classique en ajoutant le mot clé static.

Exemple (java 1.5):

```
import static java.lang.Math.*;

public class TestStaticImport {

  public static void main(String[] args) {
    System.out.println(PI);
    System.out.println(sin(0));
  }

}
```

L'utilisation de l'importation statique s'applique à tous les membres statiques : constantes et méthodes statiques de l'élément importé.

9.4. Les méta données (Meta Data)

Cette nouvelle fonctionnalité est spécifiée dans la JSR 175.

Elle propose de standardiser l'ajout d'annotations dans le code. Ces annotations pourront ensuite être traitées par des outils pour générer d'autres éléments tel que des fichiers de configuration ou du code source.

Ces annotations concernent les classes, les méthodes et les champs. Leurs syntaxes utilisent le caractère « @ ».

9.5. Les arguments variables (varargs)

Cette nouvelle fonctionnalité va permettre de passer un nombre non défini d'arguments d'un même type à une méthode. Ceci va éviter de devoir encapsuler ces données dans une collection.

Cette nouvelle fonctionnalité est spécifiée dans la JSR 201. Elle implique une nouvelle notation pour préciser la répétition d'un type d'argument. Cette nouvelle notation utilise trois petits points : ...

Exemple (java 1.5):

```
public class TestVarargs {

  public static void main(String[] args) {
    System.out.println("valeur 1 = " + additionner(1,2,3));
    System.out.println("valeur 2 = " + additionner(2,5,6,8,10));
  }

  public static int additionner(int ... valeurs) {
        int total = 0;

        for (int val : valeurs) {
                total += val;
        }

        return total;
  }

}
```

Résultat :

```
C:\tiger>java TestVarargs
valeur 1 = 6
valeur 2 = 31
```

L'utilisation de la notation ... permet le passage d'un nombre indéfini de paramètres du type précisé. Tous ces paramètres sont traités comme un tableau : il est d'ailleurs possible de fournir les valeurs sous la forme d'un tableau.

Exemple (java 1.5):

```
public class TestVarargs2 {

  public static void main(String[] args) {
    int[] valeurs = {1,2,3,4};
    System.out.println("valeur 1 = " + additionner(valeurs));
  }

  public static int additionner(int ... valeurs) {
        int total = 0;

        for (int val : valeurs) {
                total += val;
        }

        return total;
  }

}
```

Résultat :

```
C:\tiger>java TestVarargs2
valeur 1 = 10
```

Il n'est cependant pas possible de mixer des éléments unitaires et un tableau dans la liste des éléments fournis en paramètres.

Exemple (java 1.5):

```
public class TestVarargs3 {

  public static void main(String[] args) {
    int[] valeurs = {1,2,3,4};
    System.out.println("valeur 1 = " + additionner(5,6,7,valeurs));
  }

  public static int additionner(int ... valeurs) {
        int total = 0;

        for (int val : valeurs) {
                total += val;
        }

        return total;
  }

}
```

Résultat :

```
C:\tiger>javac -source 1.5 -target 1.5 TestVarargs3.java
```

```
TestVarargs3.java:7: additionner(int[]) in TestVarargs3 cannot be applied to (in
t,int,int,int[])
    System.out.println("valeur 1 = " + additionner(5,6,7,valeurs));
                                       ^
1 error
```

9.6. Les generics

Les generics permettent d'accroître la lisibilité du code et surtout de renforcer la sécurité du code grâce à un renforcement du typage. Ils permettent de préciser explicitement le type d'un objet et rendent le cast vers ce type implicite. Cette nouvelle fonctionnalité est spécifiée dans la JSR 14.

Ils permettent par exemple de spécifier quel type d'objets une collection peut contenir et ainsi éviter l'utilisation d'un cast pour obtenir un élément de la collection.

L'inconvénient majeur du cast est que celui−ci ne peut être vérifié qu'à l'exécution et qu'il peut échouer. Avec l'utilisation des generics, le compilateur pourra réaliser cette vérification lors de la phase de compilation : la sécurité du code est ainsi renforcée.

Exemple (java 1.5):

```
import java.util.*;

public class TestGenericsOld {

  public static void main(String[] args) {

    List liste = new ArrayList();
    String valeur = null;
    for(int i = 0; i < 10; i++) {
      valeur = ""+i;
      liste.add(valeur);
    }

    for (Iterator iter = liste.iterator(); iter.hasNext(); ) {
      valeur = (String) iter.next();
      System.out.println(valeur.toUpperCase());
    }
  }

}
```

L'utilisation des generics va permettre au compilateur de faire la vérification au moment de la compilation est de s'assurer ainsi qu'elle s'exécutera correctement. Ce mécanisme permet de s'assurer que les objets contenus dans la collection seront homogènes.

La syntaxe pour mettre en oeuvre les generics utilise les symboles < et > pour préciser le ou les types des objets à utiliser. Seuls des objets peuvent être utilisés avec les generics : si un type primitif est utilisé dans les generics, une erreur de type « unexpected type » est générée lors de la compilation.

Exemple (java 1.5):

```
import java.util.*;

public class TestGenerics {

  public static void main(String[] args) {

    List<String> liste = new ArrayList();
    String valeur = null;
    for(int i = 0; i < 10; i++) {
      valeur = ""+i;
      liste.add(valeur);
```

```
    }

    for (Iterator<String> iter = liste.iterator(); iter.hasNext(); ) {
      System.out.println(iter.next().toUpperCase());
    }
  }

}
```

Si un objet de type différent de celui déclaré dans le generics est utilisé dans le code, le compilateur émet une erreur lors de la compilation.

Exemple (java 1.5):

```
import java.util.*;

public class TestGenerics2 {

  public static void main(String[] args) {

    List<String> liste = new ArrayList();
    String valeur = null;
    for(int i = 0; i < 10; i++) {
      valeur = new Date();
      liste.add(valeur);
    }

    for (Iterator<String> iter = liste.iterator(); iter.hasNext(); ) {
      System.out.println(iter.next().toUpperCase());
    }
  }

}
```

Résultat :

```
C:\tiger>javac -source 1.5 -target 1.5 TestGenerics2.java
TestGenerics2.java:10: incompatible types
found   : java.util.Date
required: java.lang.String
      valeur = new Date();
               ^
Note: TestGenerics2.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
```

L'utilisation des generics permet de rendre le code plus lisible et plus sûre notamment car il n'est plus nécessaire d'utiliser un cast et de définir une variable intermédiaire.

Les generics peuvent être utilisés avec trois éléments :

- Les classes
- Les interfaces
- Les méthodes

Pour définir une classe utilisant les generics, il suffit de déclarer leur utilisation dans la signature de la classe à l'aide des caractères < et >. Ce type de déclaration est appelé type paramétré (parameterized type) Dans ce cas, les paramètres fournis dans la déclaration du generics sont des variables de types. Si la déclaration possède plusieurs variables de type alors il faut les séparer par un caractère virgule.

Exemple (java 1.5):

```
public class MaClasseGeneric<T1, T2> {
  private T1 param1;
  private T2 param2;
```

```
  public MaClasseGeneric(T1 param1, T2 param2) {
    this.param1 = param1;
    this.param2 = param2;
  }

  public T1 getParam1() {
    return this.param1;
  }

  public T2 getParam2() {
    return this.param2;
  }
}
```

Lors de l'utilisation de la classe, il faut utiliser les types paramétrés pour indiquer le type des objets à utiliser.

Exemple (java 1.5):

```
import java.util.*;

public class TestClasseGeneric {

  public static void main(String[] args) {
    MaClasseGeneric<Integer, String> maClasse =
      new MaClasseGeneric<Integer, String>(1, "valeur 1");
    Integer param1 = maClasse.getParam1();
    String  param2 = maClasse.getParam2();

  }

}
```

Le principe est identique avec les interfaces.

La syntaxe utilisant les caractères < et > se situe toujours après l'entité qu'elle concerne

Exemple (java 1.5):

```
   MaClasseGeneric<Integer, String> maClasse =
      new MaClasseGeneric<Integer, String>(1, "valeur 1");
    MaClasseGeneric<Integer, String>[] maClasses;
```

Même le cast peut être utilisé avec le generics en utilisant le nom du type paramétré dans le cast.

Il est possible de préciser une relation entre une variable de type et une classe ou interface : ainsi il sera possible d'utiliser une instance du type paramétré avec n'importe quel objet qui hérite ou implémente la classe ou l'interface précisée avec le mot clé extend dans la variable de type.

Exemple (java 1.5):

```
import java.util.*;

public class MaClasseGeneric2<T1 extends Collection> {
  private T1 param1;

  public MaClasseGeneric2(T1 param1) {
    this.param1 = param1;
  }

  public T1 getParam1() {
    return this.param1;
  }

}
```

L'utilisation du type paramétré MaClasseGeneric2 peut être réalisée avec n'importe quelle classe qui hérite de l'interface java.util.Collection.

Exemple (java 1.5):

```
import java.util.*;

public class TestClasseGeneric2 {

  public static void main(String[] args) {
    MaClasseGeneric2<ArrayList> maClasseA =
      new MaClasseGeneric2<ArrayList>(new ArrayList());
    MaClasseGeneric2<TreeSet> maClasseB =
      new MaClasseGeneric2<TreeSet>(new TreeSet());
  }

}
```

Ce mécanisme permet une utilisation un peu moins strict du typage dans les generics.

L'utilisation d'une classe qui n'hérite pas de la classe où n'implémente pas l'interface définie dans la variable de type, provoque une erreur à la compilation.

Exemple (java 1.5):

```
C:\tiger>javac −source 1.5 −target 1.5 TestClasseGeneric2.java
TestClasseGeneric2.java:8: type parameter java.lang.String is not within its bou
nd
    MaClasseGeneric2<String> maClasseC = new MaClasseGeneric2<String>("test");
                     ^
TestClasseGeneric2.java:8: type parameter java.lang.String is not within its bou
nd
    MaClasseGeneric2<String> maClasseC = new MaClasseGeneric2<String>("test");
                                                                ^
2 errors
```

9.7. Amélioration des boucles pour les collections

L'itération sur les éléments d'une collection est fastidieuse avec la déclaration d'un objet de type Iterator.

Exemple :

```
import java.util.*;

public class TestForOld {

  public static void main(String[] args) {
    List liste = new ArrayList();
    for(int i = 0; i < 10; i++) {
      liste.add(i);
    }

    for (Iterator iter = liste.iterator(); iter.hasNext(); ) {
      System.out.println(iter.next());
    }
  }

}
```

La nouvelle forme de l'instruction for, spécifiée dans la JSR 201, permet de simplifier l'écriture du code pour réaliser une telle itération et laisse le soin au compilateur de générer le code nécessaire.

Exemple (java 1.5):

```
import java.util.*;

public class TestFor {

  public static void main(String[] args) {
    List liste = new ArrayList();
    for(int i = 0; i < 10; i++) {
      liste.add(i);
    }

    for (Object element : liste) {
      System.out.println(element);
    }
  }

}
```

L'utilisation de la nouvelle syntaxe de l'instruction for peut être renforcée en combinaison avec les generics, ce qui évite l'utilisation d'un cast.

Exemple (java 1.5):

```
import java.util.*;
import java.text.*;

public class TestForGenerics {

  public static void main(String[] args) {
    List<Date> liste = new ArrayList();
    for(int i = 0; i < 10; i++) {
      liste.add(new Date());
    }

    DateFormat df = DateFormat.getDateInstance();

    for (Date element : liste) {
      System.out.println(df.format(element));
    }
  }

}
```

La nouvelle syntaxe de l'instruction peut aussi être utilisée pour parcourir tous les éléments d'un tableau.

Exemple (java 1.5):

```
import java.util.*;

public class TestForArray {

  public static void main(String[] args) {
    int[] tableau = {0,1,2,3,4,5,6,7,8,9};

    for (int element : tableau) {
      System.out.println(element);
    }
  }

}
```

L'exemple précédent fait aussi usage d'une autre nouvelle fonctionnalité du JDK 1.5 : l'unboxing.

Cela permet d'éviter la déclaration et la gestion dans le code d'une variable contenant l'index courant lors du parcours du tableau.

9.8. Les énumérations (type enum)

Souvent lors de l'écriture de code, il est utile de pouvoir définir un ensemble fini de valeurs pour une donnée pour par exemple définir les valeurs possibles qui vont caractériser l'état de cette donnée.

Pour cela, le type enum permet de définir un ensemble de constantes. Cette fonctionnalité existe déjà dans les langages C et Delphi entre autre.

Cette nouvelle fonctionnalité est spécifiée dans la JSR 201.

Jusqu'à la version 1.4, la façon la plus pratique pour palier au manque du type enum était de créer des constantes dans une classe.

Exemple (java 1.5):

```
public class MonStyle {
  public static final int STYLE_1   = 1;
  public static final int STYLE_2   = 2;
  public static final int STYLE_3   = 3;
  public static final int STYLE_4   = 4;
  public static final int STYLE_5   = 5;
}
```

Le principal inconvénient de cette technique est qu'il n'y a pas de contrôle sur la valeur affectée à une donnée surtout si les constantes ne sont pas utilisées.

La version 1.5 propose une fonctionnalité pour déclarer et utiliser un type enumération qui repose sur trois éléments :

- le mot clé enum
- un nom pour désigner l'énumération
- un ensemble de valeurs séparées par des virgules

Exemple (java 1.5):

```
public enum MonStyle { STYLE_1, STYLE_2, STYLE_3, STYLE_4, STYLE_5};
```

A la rencontre de mot clé enum, le compilateur va automatiquement créer une classe possédant les caractéristiques suivantes :

- un champ static est défini pour chaque élément précisé dans la déclaration enum
- une méthodes values() qui renvoie un tableau avec les différents éléments définis
- une méthode valuesOf()
- la classe implémente les interface Comparable et Serialisable
- les méthodes toString(), equals(), hasCode() et CompareTo() sont redéfinies

Il est possible d'utiliser toutes ces caractéristiques.

Exemple (java 1.5):

```
public class TestEnum2 {


  public enum MonStyle { STYLE_1, STYLE_2, STYLE_3, STYLE_4, STYLE_5};

  public static void main(String[] args) {
    afficher(TestEnum.MonStyle.STYLE_2);
  }

  public static void afficher(TestEnum.MonStyle style) {
    switch(style) {
```

```
      case STYLE_1 :
        System.out.println("STYLE_1");
        break;
      case STYLE_2 :
        System.out.println("STYLE_2");
        break;
      case STYLE_3 :
        System.out.println("STYLE_3");
        break;
      case STYLE_4 :
        System.out.println("STYLE_4");
        break;
      case STYLE_5 :
        System.out.println("STYLE_5");
        break;
    }
  }
}
```

Résultat :

```
C:\tiger>javac −source 1.5 −target 1.5 TestEnum2.java

C:\tiger>java TestEnum2
STYLE_2
```

Lors de la compilation de cet exemple, une classe interne est créée pour encapsuler l'énumération.

Pour pouvoir utiliser facilement une énumération, il est possible de la définir comme une entité indépendante.

Exemple (java 1.5) : le fichier MonStyle.java

```
public enum MonStyle { STYLE_1, STYLE_2, STYLE_3, STYLE_4, STYLE_5};
```

Une fois définie, il est possible d'utiliser l'énumération simplement en définissant une variable du type de l'énumération

Exemple (java 1.5):

```
public class TestEnum3 {

  private String nom;
  private MonStyle style;

  public TestEnum3(String nom, MonStyle style) {
    this.nom = nom;
    this.style = style;
  }

  public static void main(String[] args) {
    TestEnum3 te = new TestEnum3("objet1",MonStyle.STYLE_1);
  }
}
```

Les énumérations étant transformées en une classe par le compilateur, il y a une vérification de type faite de l'utilisation de l'énumération à la compilation.

La classe générée possède les caractéristiques suivantes :

- hérite de la classe java.lang.Enum
- déclarée final pour empêcher toute modification sur la classe par héritage
- il n'y a pas de constructeur public
- chacune des valeurs de l'énumération est une instance de la classe encapsulant l'énumération
- chacune des valeurs est déclarée avec les modificateurs public, static et final ce qui les rend non modifiables
- les valeurs peuvent être testées avec l'opérateur == puisqu'elles sont déclarées avec le modificateur final

- la méthode valueOf() est définie
- la méthode values() renvoie un tableau des valeurs de l'énumération sous la forme

L'instruction switch a été modifiée pour permettre de l'utiliser avec une énumération puisque bien qu'étant physiquement une classe, celle−ci possède une liste finie de valeurs associées.

Remarque : dans les différents cas de l'instruction switch, il n'est pas utile de préfixer chaque valeur de l'énumération utilisée par le nom de l'énumération puisque celle ci est automatiquement déterminée par le compilateur à partir de la variable passée en paramètre de l'instruction switch.

Exemple (java 1.5):

```
public class TestEnum4 {

  private String nom;
  private MonStyle style;

  public TestEnum4(String nom, MonStyle style) {
    this.nom = nom;
    this.style = style;
  }

  private void afficher() {
    switch(style) {
      case STYLE_1:
        System.out.println("Style numero 1");
        break;
      case STYLE_2:
        System.out.println("Style numero 2");
        break;
      case STYLE_3:
        System.out.println("Style numero 3");
        break;
      case STYLE_4:
        System.out.println("Style numero 4");
        break;
      case STYLE_5:
        System.out.println("Style numero 5");
        break;
      default:
        System.out.println("Style inconnu");
        break;
    }
  }

  public static void main(String[] args) {
    TestEnum4 te = new TestEnum4("objet1",MonStyle.STYLE_1);
    te.afficher();
  }
}
```

Résultat :

```
C:\tiger>java TestEnum4
Style numero 1
```

Il est possible d'associer une énumération avec un objet dans une collection de type Map en utilisant la classe EnumMap avec les generics

Exemple (java 1.5):

```
public class TestEnum5 {

  private String nom;
  private MonStyle style;

  public static void main(String[] args) {
    EnumMap<MonStyle, String> libelles = new EnumMap<MonStyle, String>(MonStyle.class);
```

```
    libelles.put(MonStyle.STYLE_1, "Libelle du style numero 1");
    libelles.put(MonStyle.STYLE_2, "Libelle du style numero 2");
    libelles.put(MonStyle.STYLE_3, "Libelle du style numero 3");
    libelles.put(MonStyle.STYLE_4, "Libelle du style numero 4");
    libelles.put(MonStyle.STYLE_5, "Libelle du style numero 5");

    System.out.println(libelles.get(MonStyle.STYLE_1));
  }
}
```

Résultat:

```
C:\tiger>java TestEnum5
Libelle du style numero 1
```

Partie 2 : Développement des interfaces graphiques

Les interfaces graphiques assurent le dialogue entre les utilisateurs et une application.

Dans un premier temps, Java propose l'API AWT pour créer des interfaces graphiques. Depuis, Java propose une nouvelle API nommée Swing. Ces deux API peuvent être utilisées pour développer des applications ou des applets. Face aux problèmes de performance de Swing, IBM a développé sa propre bibliothèque nommée SWT utilisée pour développé l'outil Eclipse. La vélocité de cette application favorise une utilisation grandissante de cette bibliothèque.

Cette partie contient les chapitres suivants :

- Le graphisme : entame une série de chapitres sur les l'interfaces graphiques en détaillant les objets et méthodes de base pour le graphisme
- Les éléments d'interfaces graphiques de l'AWT : recense les différents composants qui sont fournis dans la bibliothèque AWT
- La création d'interfaces graphiques avec AWT : indique comment réaliser des interfaces graphiques avec l'AWT
- L'interception des actions de l'utilisateur : détaille les mécanismes qui permettent de réagir aux actions de l'utilisateur via une interface graphique
- Le développement d'interfaces graphiques avec SWING : indique comment réaliser des interfaces graphiques avec Swing
- Le développement d'interfaces graphiques avec SWT : indique comment réaliser des interfaces graphiques avec SWT
- JFace
- Les applets : plonge au coeur des premières applications qui ont rendu Java célèbre

10. Le graphisme

La classe Graphics contient les outils nécessaires pour dessiner. Cette classe est abstraite et elle ne possède pas de constructeur public : il n'est pas possible de construire des instances de graphics nous même. Les instances nécessaires sont fournies par le système d'exploitation qui instanciera via à la machine virtuelle une sous classe de Graphics dépendante de la plateforme utilisée.

Ce chapitre contient plusieurs sections :

10.1. Les opérations sur le contexte graphique

10.1.1. Le tracé de formes géométriques

A l 'exception des lignes, toutes les formes peuvent être dessinées vides (méthode drawXXX) ou pleines (fillXXX).

La classe Graphics possède de nombreuses méthodes qui permettent de réaliser des dessins.

Méthode	Role
drawRect(x, y, largeur, hauteur) fillRect(x, y, largeur, hauteur)	dessiner un carré ou un rectangle
drawRoundRect(x, y, largeur, hauteur, hor_arr,ver_arr) fillRoundRect(x, y, largeur, hauteur, hor_arr, ver_arr)	dessiner un carré ou un rectangle arrondi
drawLine(x1, y1, x2, y2)	Dessiner une ligne
drawOval(x, y, largeur, hauteur) fillOval(x, y, largeur, hauteur)	dessiner un cercle ou une elipse en spécifiant le rectangle dans lequel ils s'incrivent

drawPolygon(int[], int[], int)
fillPolygon(int[], int[], int)
	Dessiner un polygone ouvert ou fermé. Les deux premiers paramètres sont les coordonnées en abscisses et en ordonnées. Le dernier paramètre est le nombre de points du polygone.
	Pour dessiner un polygone fermé il faut joindre le dernier point au premier.
	Exemple (code Java 1.1) :
	`int[] x={10,60,100,80,150,10};`
	`int[] y={15,15,25,35,45,15};`

	`g.drawPolygon(x,y,x.length);`
	`g.fillPolygon(x,y,x.length);`
	
	Il est possible de définir un objet Polygon.
	
	Exemple (code Java 1.1) :
	`int[] x={10,60,100,80,150,10};`
	`int[] y={15,15,25,35,45,15};`
	`Polygon p = new Polygon(x, y,x.length );`
	`g.drawPolygon(p);`

drawArc(x, y, largeur, hauteur, angle_deb, angle_bal) fillArc(x, y, largeur, hauteur, angle_deb, angle_bal);	dessiner un arc d'ellipse inscrit dans un rectangle ou un carré. L'angle 0 se situe à 3 heures. Il faut indiquer l'angle de début et l'angle balayé

10.1.2. Le tracé de texte

La méthode drawString() permet d'afficher un texte aux coordonnées précisées

Exemple (code Java 1.1) :

```
g.drawString(texte, x, y );
```

Pour afficher des nombres int ou float, il suffit de les concatener à une chaîne éventuellement vide avec l'opérateur +.

10.1.3. L'utilisation des fontes

La classe Font permet d'utiliser une police de caractères particulière pour affiche un texte.

Exemple (code Java 1.1) :

```
Font fonte = new Font(« TimesRoman »,Font.BOLD,30);
```

Le constructeur de la classe Font est Font(String, int, int). Les paramètres sont : le nom de la police, le style (BOLD, ITALIC, PLAIN ou 0,1,2) et la taille des caractères en points.

Pour associer plusieurs styles, il suffit de les additionner

Exemple (code Java 1.1) :

```
Font.BOLD + Font.ITALIC
```

Si la police spécifiée n'existe pas, Java prend la fonte par défaut même si une autre a été spécifiée précedemment. Le style et la taille seront tout de même adpatés. La méthode getName() de la classe Font retourne le nom de la fonte.

La méthode setFont() de la classe Graphics permet de changer la police d'affichage des textes

Exemple (code Java 1.1) :

```
Font fonte = new Font("
TimesRoman ",Font.BOLD,30);
g.setFont(fonte);
```

```
g.drawString("bonjour",50,50);
```

Les polices suivantes sont utilisables : Dialog, Helvetica, TimesRoman, Courier, ZapfDingBats

10.1.4. La gestion de la couleur

La méthode setColor() permet de fixer la couleur des éléments graphiques des objets de type Graphics créés après à son appel.

Exemple (code Java 1.1) :

```
g.setColor(Color.black); //(green, blue, red, white, black, ...)
```

10.1.5. Le chevauchement de figures graphiques

Si 2 surfaces de couleur différentes se superposent, alors la derniere dessinée recouvre la précédente sauf si on invoque la méthode setXORMode(). Dans ce cas, la couleur de l'intersection prend une autre couleur. L'argument à fournir est une couleur alternative. La couleur d'intersection représente une combinaison de la couleur originale et de la couleur alternative.

10.1.6. L'effacement d'une aire

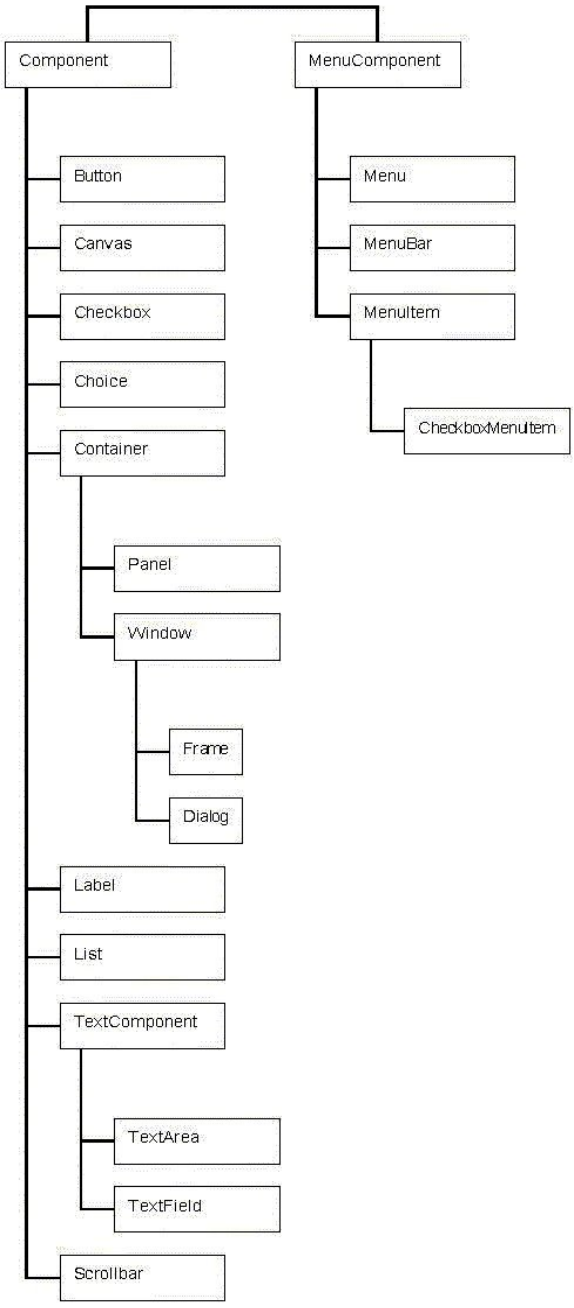
La méthode clearRect(x1, y1, x2, y2) dessine un rectangle dans la couleur de fond courante.

10.1.7. La copier une aire rectangulaire

La méthode copyArea(x1, y1, x2, y2, dx, dy) permet de copier une aire rectangulaire. Les paramètres dx et dy permettent de spécifier un décalage en pixels de la copie par rapport à l'originale.

11. Les éléments d'interfaces graphiques de l'AWT

Les classes du toolkit AWT (Abstract Windows Toolkit) permettent d'écrire des interfaces graphiques indépendantes du système d'exploitation sur lesquelles elles vont fonctionner. Cette librairie utilise le système graphique de la plateforme d'exécution (Windows, MacOS, X−Window) pour afficher les objets graphiques. Le toolkit contient des classes décrivant les composants graphiques, les polices, les couleurs et les images.



Le diagramme ci dessus définit une vue partielle de la hiérarchie des classes (les relations d'héritage) qu'il ne faut pas confondre avec la hiérarchie interne à chaque application qui définit l'imbrication des différents composants graphiques.

Les deux classes principales de AWT sont Component et Container. Chaque type d'objet de l'interface graphique est une classe dérivée de Component. La classe Container, qui hérite de Component est capable de contenir d'autres objets graphiques (tout objet dérivant de Component).

Ce chapitre contient plusieurs sections :

11.1. Les composants graphiques

Pour utiliser un composant, il faut créer un nouvel objet représentant le composant et l'ajouter à un de type conteneur qui existe avec la méthode add().

Exemple (code Java 1.1) : ajout d'un bouton dans une applet (Applet hérite de Panel)

```
import java.applet.*;
import java.awt.*;

public class AppletButton extends Applet {

   Button b = new Button(" Bouton ");

   public void init() {
      super.init();
      add(b);
   }
}
```

11.1.1. Les étiquettes

Il faut utiliser un objet de la classe java.awt.Label

Exemple (code Java 1.1) :

```
Label la = new Label( );
la.setText("une etiquette");
// ou Label la = new Label("une etiquette");
```

Il est possible de créer un objet de la classe java.awt.Label en précisant l'alignement du texte

Exemple (code Java 1.1) :

```
Label la = new Label("etiquette", Label.RIGHT);
```

Le texte à afficher et l'alignement peuvent être modifiés dynamiquement lors de l'éxecution :

Exemple (code Java 1.1) :

```
la.setText("nouveau texte");
la.setAlignment(Label.LEFT);
```

11.1.2. Les boutons

Il faut utiliser un objet de la classe java.awt.Button

Cette classe possède deux constructeurs :

Constructeur	Rôle
Button()	
Button(String)	Permet de préciser le libellé du bouton

Exemple (code Java 1.1) :

```
Button bouton = new Button();
bouton.setLabel("bouton");
// ou Button bouton = new Button("bouton");
```

Le libellé du bouton peut être modifié dynamiquement grace à la méthode setLabel() :

Exemple (code Java 1.1) :

```
bouton.setLabel("nouveau libellé");
```

11.1.3. Les panneaux

Les panneaux sont des conteneurs qui permettent de rassembler des composants et de les positionner grace à un gestionnaire de présentation. Il faut utiliser un objet de la classe java.awt.Panel.

Par défaut le gestionnaire de présentation d'un panel est de type FlowLayout.

Constructeur	Role
Panel()	Créer un panneau avec un gestionnaire de présentation de type FlowLayout
Panel(LayoutManager)	Créer un panneau avec le gestionnaire précisé en paramètre

Exemple (code Java 1.1) :

```
Panel p = new Panel();
```

L'ajout d'un composant au panel se fait grace à la méthode add().

Exemple (code Java 1.1) :

```
p.add(new Button("bouton");
```

11.1.4. Les listes déroulantes (combobox)

Il faut utiliser un objet de la classe java.awt.Choice

Cette classe ne possède qu'un seul constructeur qui ne possèdent pas de paramètres.

Exemple (code Java 1.1) :

```
Choice maCombo = new Choice();
```

Les méthodes add() et addItem() permettent d'ajouter des éléments à la combo.

Exemple (code Java 1.1) :

```
maCombo.addItem("element 1");
// ou maCombo.add("element 2");
```

Plusieurs méthodes permettent la gestion des sélections :

Méthodes	Role
void select(int);	sélectionner un élément par son indice : le premier élément correspond à l'indice 0. Une exception IllegalArgumentException est levée si l'indice ne correspond pas à un élément. Exemple (code Java 1.1) : `maCombo.select(0);`
void select(String);	sélectionner un élément par son contenu Aucune exception est levée si la chaîne de caractères ne correspond à aucun élément : l'élément sélectionné ne change pas. Exemple (code Java 1.1) : `maCombo.select("element 1");`

int countItems( );	déterminer le nombre d'élément de la liste. La méthode countItems() permet d'obtenir le nombre d'éléments de la combo.
	
	Exemple (code Java 1.1) :
	`int n;`
	`n=maCombo.countItems();`
	
	 
	
	il faut utiliser getItemCount() à la place
	
	Exemple (code Java 1.1) :
	`int n;`
	`n=maCombo.getItemCount();`

String getItem(int);	lire le contenu de l'élément d'indice n
	
	Exemple (code Java 1.1) :
	`String c = new String( );`
	`c = maCombo.getItem(n);`

String getSelectedItem();	déterminer le contenu de l'élément sélectionné
	
	Exemple (code Java 1.1) :
	`String s = new String( );`
	`s = maCombo.getSelectedItem();`

int getSelectedIndex( );	déterminer l'index de l'élément selectionné
	Exemple (code Java 1.1) :
	`int n;`
	`n=maCombo.getSelectedIndex();`

11.1.5. La classe TextComponent

La classe TextComponent est la classe des mères des classes qui permettent l'édition de texte : TextArea et TextField.

Elle définit un certain nombre de méthodes dont ces classes héritent.

Méthodes	Role
String getSelectedText( );	Renvoie le texte sélectionné
int getSelectionStart( );	Renvoie la position de début de sélection
int getSelectionEnd( );	Renvoie la position de fin de sélection
String getText( );	Renvoie le texte contenu dans l'objet
boolean isEditable( );	Retourne un booleen indiquant si le texte est modifiable
void select(int start, int end);	Sélection des caractères situés entre start et end
void selectAll( );	Sélection de tout le texte
void setEditable(boolean b);	Autoriser ou interdire la modification du texte
void setText(String s);	Définir un nouveau texte

11.1.6. Les champs de texte

Il faut déclarer un objet de la classe java.awt.TextField

Il existe plusieurs constructeurs :

Constructeurs	Role
TextField();	
TextField(int);	prédetermination du nombre de caractères à saisir
TextField(String);	avec texte par défaut
TextField(String, int);	avec texte par défaut et nombre de caractères à saisir

Cette classe possède quelques méthodes utiles :

Méthodes	Role

String getText( )	lecture de la chaîne saisie
	Exemple (code Java 1.1) :
	`String saisie = new String();`
	`saisie = tf.getText( );`

int getColumns( )	lecture du nombre de caractères predéfini

	Exemple (code Java 1.1) :
	`int i;`
	`i = tf.getColumns( );`

void setEchoCharacter()	pour la saisie d'un mot de passe : remplace chaque caractère saisi par celui fourni en paramètre
	Exemple (code Java 1.1) :
	`tf.setEchoCharacter('*');`
	`TextField tf = new TextField(10);`
	 
	
	il faut utiliser la méthode setEchoChar()
	Exemple (code Java 1.1) :
	`tf.setEchoChar('*');`

11.1.7. Les zones de texte multilignes

Il faut déclarer un objet de la classe java.awt.TextArea

Il existe plusieurs constructeurs :

Constructeur	Role
TextArea()	
TextArea(int, int)	avec prédetermination du nombre de lignes et de colonnes
TextArea(String)	avec texte par défaut
TextArea(String, int, int)	avec texte par défaut et taille

Les principales méthodes sont :

Méthodes	Role

String getText()	lecture du contenu intégral de la zone de texte
	Exemple (code Java 1.1) :
	`String contenu = new String;`
	`contenu = ta.getText( );`

String getSelectedText( )	lecture de la portion de texte sélectionnée
	Exemple (code Java 1.1) :
	`String contenu = new String;`
	`contenu = ta.getSelectedText( );`

int getRows()	détermination du nombre de lignes
	Exemple (code Java 1.1) :
	`int n;`
	`n = ta.getRows( );`

int getColumns( )	détermination du nombre de colonnes
	Exemple (code Java 1.1) :
	`int n;`
	`n = ta.getColumns( );`

void insertText(String, int)	insertion de la chaîne à la position fournie
	Exemple (code Java 1.1) :
	`String text =`
	`new String(«texte inséré»);`
	`int n =10;`
	`ta.insertText(text,n);`
	 
	Il faut utiliser la méthode insert()
	Exemple (code Java 1.1) :
	`String text = new`
	`String(«texte inséré»);`
	`int n =10;`
	`ta.insert(text,n);`

void setEditable(boolean)	Autoriser la modification
	Exemple (code Java 1.1) :
	`ta.setEditable(False);`
	`//texte non modifiable`

void appendText(String)	Ajouter le texte transmis au texte existant Exemple (code Java 1.1) : `ta.appendTexte(String text);`  Il faut utiliser la méthode append()
void replaceText(String, int, int)	Remplacer par text le texte entre les positions start et end Exemple (code Java 1.1) : `ta.replaceText(text, 10, 20);`  il faut utiliser la méthode replaceRange()

11.1.8. Les listes

Il faut déclarer un objet de la classe java.awt.List.

Il existe plusieurs constructeurs :

Constructeur	Role
List( )	
List(int)	Permet de préciser le nombre de lignes affichées

List(int, boolean)	Permet de préciser le nombre de lignes affichées et l'indicateur de sélection multiple

Les principales méthodes sont :

Méthodes	Role

void addItem(String)	ajouter un élément
	Exemple (code Java 1.1) :
	`li.addItem("nouvel element");`
	`// a jout en fin de liste`
	  il faut utiliser la méthode add()

void addItem(String, int)	insérer un élément à un certain emplacement : le premier element est en position 0 Exemple (code Java 1.1) : `li.addItem("ajout ligne",2);`  il faut utiliser la méthode add()

void delItem(int)	retirer un élément de la liste
	Exemple (code Java 1.1) :
	`li.delItem(0);`
	`// supprime le premier element`
	  il faut utiliser la méthode remove()

void delItems(int, int)	supprimer plusieurs éléments consécutifs entre les deux indices Exemple (code Java 1.1) : `li.delItems(1, 3);`  cette méthode est deprecated
void clear()	effacement complet du contenu de la liste Exemple (code Java 1.1) : `li.clear( );`  il faut utiliser la méthode removeAll()

void replaceItem(String, int)	remplacer un élément
	Exemple (code Java 1.1) :
	`li.replaceItem(`
	`"ligne remplacee", 1);`

int countItems()	nombre d'élément de la liste
	Exemple (code Java 1.1) :
	`int n;`
	`n = li.countItems( );`
	  il faut utiliser la méthode getItemCount()

int getRows()	nombre de ligne de la liste
	Exemple (code Java 1.1) :
	`int n;`
	`n = li.getRows( );`

String getItem(int)	contenu d'un élément
	Exemple (code Java 1.1) :
	`String text = new String( );`
	`text = li.getItem(1);`

void select(int)	sélectionner un élément Exemple (code Java 1.1) : `li.select(0);`
setMultipleSelections(boolean)	déterminer si la sélection multiple est autorisée Exemple (code Java 1.1) : `li.setMultipleSelections(true);`  il faut utiliser la méthode setMultipleMode()
void deselect(int)	désélectionner un élément Exemple (code Java 1.1) : `li.deselect(0);`

int getSelectedIndex( )	déterminer l'élément sélectionné en cas de selection simple : renvoie l'indice ou −1 si aucun élément n'est sélectionné
	Exemple (code Java 1.1) :
	`int i;`
	`i = li.getSelectedIndex();`

int[] getSelectedIndexes( )	déterminer les éléments sélectionnées en cas de sélection multiple Exemple (code Java 1.1) : `int i[]=li.getSelectedIndexes();`
String getSelectedItem( )	déterminer le contenu en cas de sélection simple : renvoie le texte ou null si pas de sélection Exemple (code Java 1.1) :

	`String texte = new String( );` `texte = li.getSelectedItem( );`

String[] getSelectedItems()	déterminer les contenus des éléments sélectionnés en cas de sélection multiple : renvoie les textes sélectionnés ou null si pas de sélection
	Exemple (code Java 1.1) :
	`String texte[ ] =`
	`li.getSelectedItems();`
	`for (i = 0 ; i <`
	`texte.length(); i++)`
	`  System.out.println(texte[i]);`

boolean isSelected(int)	déterminer si un élément est sélectionné
	Exemple (code Java 1.1) :
	`boolean selection;`
	`selection = li.isSelected(0);`
	  il faut utiliser la méthode isIndexSelect()

int getVisibleIndex()	renvoie l'index de l'entrée en haut de la liste Exemple (code Java 1.1) : `int top = li.getVisibleIndex();`
void makeVisible(int)	assure que l'élement précisé sera visible Exemple (code Java 1.1) : `li.makeVisible(10);`

Exemple (code Java 1.1) : une liste de 5 éléments avec sélection multiple

```
import java.awt.*;

class TestList {

  static public void main (String arg [ ]) {

    Frame frame = new Frame("Une liste");

    List list = new List(5,true);
    list.add("element 0");
    list.add("element 1");
    list.add("element 2");
    list.add("element 3");
    list.add("element 4");

    frame.add(List);
    frame.show();
    frame.pack();
  }
}
```

11.1.9. Les cases à cocher

Il faut déclarer un objet de la classe java.awt.Checkbox

Il existe plusieurs constructeurs :

Constructeur	Role
Checkbox( )	
Checkbox(String)	avec une étiquette
Checkbox(String,boolean)	avec une étiquette et un état
Checkbox(String,CheckboxGroup, boolean)	avec une étiquette, dans un groupe de cases à cocher et un état

Les principales méthodes sont :

Méthodes	Role
void setLabel(String)	modifier l'étiquette Exemple (code Java 1.1) : `cb.setLabel( "libelle de la case : " );`
void setState(boolean)	fixer l'état Exemple (code Java 1.1) : `cb.setState( true );`

boolean getState( )	consulter l'état de la case
	Exemple (code Java 1.1) :
	`boolean etat;`
	`etat = cb.getState( );`

String getLabel( )	lire l'étiquette de la case
	Exemple (code Java 1.1) :
	`String commentaire = new String( );`
	`commentaire = cb.getLabel( );`

11.1.10. Les boutons radio

Déclarer un objet de la classe java.awt.CheckboxGroup

Exemple (code Java 1.1) :

```
CheckboxGroup rb;
Checkbox cb1 = new Checkbox(« etiquette 1 », rb, etat1_boolean);
Checkbox cb2 = new Checkbox(« etiquette 2 », rb, etat1_boolean);
Checkbox cb3 = new Checkbox(« etiquette 3 », rb, etat1_boolean);
```

Les principales méthodes sont :

Méthodes	Role
Checkbox getCurrent()	retourne l'objet Checkbox correspondant à la réponse sélectionnée  il faut utiliser la méthode getSelectedCheckbox()

void setCurrent(Checkbox)	Coche le bouton radio passé en paramètre  il faut utiliser la méthode setSelectedCheckbox()

11.1.11. Les barres de défilement

Il faut déclarer un objet de la classe java.awt.Scrollbar

Il existe plusieurs constructeurs :

Constructeur	Role
Scrollbar( )	
Scrollbar(orientation)	
Scrollbar(orientation, valeur_initiale, visible, min, max)	

- orientation : Scrollbar.VERTICALE ou Scrollbar.HORIZONTAL
- valeur_initiale : position du curseur à la création
- visible : taille de la partie visible de la zone défilante
- min : valeur minimale associée à la barre
- max : valeur maximale associée à la barre

Les principales méthodes sont :

Méthodes	Role

sb.setValues(int,int,int,int)	maj des parametres de la barre
	Exemple (code Java 1.1) :
	`sb.setValues(`
	`valeur, visible,`
	`minimum, maximum );`

void setValue(int)	modifer la valeur courante Exemple (code Java 1.1) : `sb.setValue(10);`

int getMaximum( );	lecture du maximum
	Exemple (code Java 1.1) :
	`int max =`
	`sb.getMaximum( );`

int getMinimum( );	lecture du minimum
	Exemple (code Java 1.1) :
	`int min =`
	`sb.getMinimum( );`

int getOrientation( )	lecture de l'orientation Exemple (code Java 1.1) : `int o =`

	`sb.getOrientation( );`

int getValue( );	lecture de la valeure courante
	Exemple (code Java 1.1) :
	`int valeur =`
	`sb.getValue( );`

void setLineIncrement(int);	détermine la valeur à ajouter ou à oter quand l'utilisateur clique sur une flèche de défilement  il faut utiliser la méthode setUnitIncrement()
int setPageIncrement( );	détermine la valeur à ajouter ou à oter quand l'utilisateur clique sur le conteneur  il faut utiliser la méthode setBlockIncrement()

11.1.12. La classe Canvas

C'est un composant sans fonction particulière : il est utile pour créer des composants graphiques personnalisés.

Il est nécessaire d'étendre la classe Canvas pour en redéfinir la méthode Paint().

syntaxe : Cancas can = new Canvas( );

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MonCanvas extends Canvas {

   public void paint(Graphics g) {
      g.setColor(Color.black);
      g.fillRect(10, 10, 100,50);
      g.setColor(Color.green);
      g.fillOval(40, 40, 10,10);
   }

}


import java.applet.*;
import java.awt.*;

public class AppletButton extends Applet {

   MonCanvas mc = new MonCanvas();

   public void paint(Graphics g) {
      super.paint(g);
      mc.paint(g);
   }
}
```

11.2. La classe Component

Les contrôles fenêtrés descendent plus ou moins directement de la classe AWT Component.

Cette classe contient de nombreuse méthodes :

Méthodes	Role
Rectangle bounds()	renvoie la position actuelle et la taille des composants  utiliser la méthode getBounds().
void disable()	désactive les composants  utiliser la méthode setEnabled(false).
void enable()	active les composants  utiliser la méthode setEnabled(true).
void enable(boolean)	active ou désactive le composant selon la valeur du paramètre  utiliser la méthode setEnabled(boolean).
Color getBackGround()	renvoie la couleur actuelle d'arrière plan
Font getFont()	renvoie la fonte utilisée pour afficher les caractères
Color getForeGround()	renvoie la couleur de premier plan
Graphics getGraphics()	renvoie le contexte graphique
Container getParent()	renvoie le conteneur (composant de niveau supérieure)
void hide()	masque l'objet  utiliser la méthode setVisible().
boolean inside(int x, int y)	indique si la coordonnée écran absolue se trouve dans l'objet  utiliser la méthode contains().
boolean isEnabled()	indique si l'objet est actif
boolean isShowing()	indique si l'objet est visible
boolean isVisible()	indique si l'objet est visible lorque sont conteneur est visible
boolean isShowing()	indique si une partie de l'objet est visible
void layout()	repositionne l'objet en fonction du Layout Manager courant  utiliser la méthode doLayout().
Component locate(int x, int y)	retourne le composant situé à cet endroit  utiliser la méthode getComponentAt().
Point location()	retourne l'origine du composant  utiliser la méthode getLocation().

void move(int x, int y)	déplace les composants vers la position spécifiée  utiliser la méthode setLocation().
void paint(Graphics);	dessine le composant
void paintAll(Graphics)	dessine le composant et ceux qui sont contenus en lui
void repaint()	redessine le composant pat appel à la méthode update()
void requestFocus();	demande le focus
void reshape(int x, inty, int w, int h)	modifie la position et la taille (unité : points écran)  utiliser la méthode setBounds().
void resize(int w, int h)	modifie la taille (unité : points écran)  utiliser la méthode setSize().
void setBackground(Color)	définit la couleur d'arrière plan
void setFont(Font)	définit la police
void setForeground(Color)	définit la couleur de premier plan
void show()	affiche le composant  utiliser la méthode setVisible(True).
Dimension size()	détermine la taille actuelle  utiliser la méthode getSize().

11.3. Les conteneurs

Les conteneurs sont des objets graphiques qui peuvent contenir d'autres objets graphiques, incluant éventuellement des conteneurs. Ils héritent de la classe Container.

Un composant graphique doit toujours être incorporé dans un conteneur :

Conteneur	Rôle
Panel	conteneur sans fenêtre propre. Utile pour ordonner les contrôles
Window	fenêtre principale sans cadre ni menu. Les objets descendants de cette classe peuvent servir à implémenter des menus
Dialog (descendant de Window)	réaliser des boîtes de dialogue simples
Frame (descendant de Window)	classe de fenêtre completement fonctionnelle
Applet (descendant de Panel)	pas de menu. Pas de boîte de dialogue sans être incorporée dans une classe Frame.

L'insertion de composant dans un conteneur se fait grace à la méthode add(Component) de la classe Container.

Exemple (code Java 1.1) :

```
Panel p = new Panel();

Button b1 = new button(« premier »);
p.add(b1);
Button b2;
p.add(b2 = new Button (« Deuxième »);
p.add(new Button(«Troisième »);
```

11.3.1. Le conteneur Panel

C'est essentiellement un objet de rangement pour d'autres composants.

La classe Panel possède deux constructeurs :

Constructeur	Role
Panel()	
Panel(LayoutManager)	Permet de préciser un layout manager

Exemple (code Java 1.1) :

```
Panel p = new Panel( );

Button b = new Button(« bouton »);
p.add( b);
```

11.3.2. Le conteneur Window

La classe Window contient plusieurs méthodes dont voici les plus utiles :

Méthodes	Role
void pack()	Calculer la taille et la position de tous les contrôles de la fenêtre. La méthode pack() agit en étroite collaboration avec le layout manager et permet à chaque contrôle de garder, dans un premier temps sa taille optimale. Une fois que tous les contrôles ont leur taille optimale, pack() utilise ces informations pour positionner les contrôles. pack() calcule ensuite la taille de la fenêtre. L'appel à pack() doit se faire à l'intérieur du constructeur de fenêtre après insertion de tous les contrôles.
void show()	Afficher la fenêtre
void dispose()	Liberer les ressources allouée à la fenêtre

11.3.3. Le conteneur Frame

Ce conteneur permet de créer des fenêtres d'encadrement. Il hérite de la classe Window qui ne s'occupe que de l'ouverture de la fenêtre. Window ne connait pas les menus ni les bordures qui sont gérés par la classe Frame. Dans une applet, elle n'apparait pas dans le navigateur mais comme une fenêtre indépendante.

Il existe deux constructeurs :

Constructeur	Role
Frame()	Exemple : Frame f = new Frame( );
Frame(String)	Precise le nom de la fenêtre Exemple : Frame f = new Frame(« titre »);

Les principales méthodes sont :

Méthodes	Role
setCursor(int)	changer le pointeur de la souris dans la fenêtre Exemple : f.setCursor(Frame.CROSSHAIR_CURSOR);  utiliser la méthode setCursor(Cursor).
int getCursorType()	déterminer la forme actuelle du curseur  utiliser la méthode getCursor().
Image getIconImage()	déterminer l'icone actuelle de la fenêtre
MenuBar getMenuBar()	déterminer la barre de menus actuelle
String getTitle()	déterminer le titre de la fenêtre
boolean isResizeable()	déteriner si la taille est modifiable
void remove(MenuComponent)	Supprimer un menu
void setIconImage(Image);	définir l'icone de la fenêtre
void setMenuBar(MenuBar)	Définir la barre de menu
void setResizeable(boolean)	définir si la taille peut être modifiée
void SetTitle(String)	définir le titre de la fenêtre

Exemple (code Java 1.1) :

```
import java.applet.*;
import java.awt.*;

public class AppletFrame extends Applet {

   Frame f;

   public void init() {
      super.init();
      // insert code to initialize the applet here
      f = new Frame("titre");
      f.add(new Label("hello "));
      f.show();
      f.setSize(300, 100);
   }
}
```

Le message « Warning : Applet window » est impossible à enlever dans la fenêtre : cela permet d'éviter la création d'une applet qui demande un mot de passe.

Le gestionnaire de mise en page par défaut d'une Frame est BorderLayout (FlowLayout pour une applet).

Exemple (code Java 1.1) : Exemple : construction d'une fenêtre simple

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
      show(); // affiche la fenetre
   }

   public static void main(String[] args) {
      new MaFrame();
   }
}
```

11.3.4. Le conteneur Dialog

La classe Dialog hérite de la classe Window.

Une boîte de dialogue doit dérivée de la Classe Dialog de package java.awt.

Un objet de la classe Dialog doit dépendre d'un objet de la classe Frame.

Exemple (code Java 1.1) :

```
import java.awt.*;
import java.awt.event.*;

public class Apropos extends Dialog {

   public APropos(Frame parent) {
      super(parent, "A propos ", true);
      addWindowListener(new
      AProposListener(this));
      setSize(300, 300);
      setResizable(False);
   }
}

class AProposListener extends WindowAdapter {

   Dialog dialogue;
   public AProposListener(Dialog dialogue) {
      this.dialogue = dialog;
   }

   public void windowClosing(WindowEvent e) {
      dialogue.dispose();
   }

}
```

L'appel du constructeur Dialog(Frame, String, Boolean) permet de créer une instance avec comme paramètres : la fenêtre à laquelle appartient la boîte de dialogue, le titre de la boîte, le caractère modale de la boîte.

La méthode dispose() de la classe Dialog ferme la boîte et libère les ressources associées. Il ne faut pas associer cette action à la méthode windowClosed() car dispose provoque l'appel de windowClosed ce qui entrainerait un appel récursif infinie.

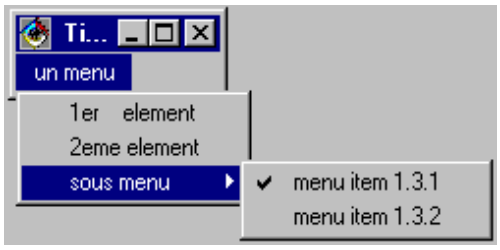
11.4. Les menus

Il faut insérer les menus dans des objets de la classe Frame (fenêtre d'encadrement). Il n'est donc pas possible d'insérer directement des menus dans une applet.

Il faut créer une barre de menu et l'affecter à la fenêtre d'encadrement. Il faut créer ensuite créer les entrées de chaque menu et les rattacher à la barre. Ajouter ensuite les éléments à chacun des menus.

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);

      MenuBar mb = new MenuBar();
      setMenuBar(mb);

      Menu m = new Menu(" un menu ");
      mb.add(m);
      m.add(new MenuItem(" 1er element "));
      m.add(new MenuItem(" 2eme element "));
      Menu m2 = new Menu(" sous menu ");

      CheckboxMenuItem cbm1 = new CheckboxMenuItem(" menu item 1.3.1 ");
      m2.add(cbm1);
      cbm1.setState(true);
      CheckboxMenuItem cbm2 = new CheckboxMenuItem(" menu item 1.3.2 ");
      m2.add(cbm2);

      m.add(m2);

      pack();
      show(); // affiche la fenetre

   }

   public static void main(String[] args) {
      new MaFrame();
   }

}
```



Exemple (code Java 1.1) : création d'une classe qui définit un menu

```
import java.awt.*;

public class MenuFenetre extends java.awt.MenuBar {

   public MenuItem menuQuitter, menuNouveau, menuApropos;

   public MenuFenetre() {

      Menu menuFichier = new Menu(" Fichier ");
      menuNouveau = new MenuItem(" Nouveau ");
      menuQuitter = new MenuItem(" Quitter ");
```

```
      menuFichier.add(menuNouveau);

      menuFichier.addSeparator();

      menuFichier.add(menuQuitter);

      Menu menuAide = new Menu(" Aide ");
      menuApropos = new MenuItem(" A propos ");
      menuAide.add(menuApropos);

      add(menuFichier);

      setHelpMenu(menuAide);
   }
}
```

La méthode setHelpMenu() confère sous certaines plateformes un comportement particulier à ce menu.

La méthode setMenuBar() de la classe Frame prend en paramètre une instance de la classe MenuBar. Cette instance peut être directement une instance de la classe MenuBar qui aura été modifiée grace aux méthodes add() ou alors une classe dérivée de MenuBar qui est adaptée aux besoins (voir Exemple);

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
      MenuFenetre mf = new
      MenuFenetre();

      setMenuBar(mf);

      pack();

      show(); // affiche la fenetre

   }

   public static void main(String[] args) {
      new MaFrame();
   }

}
```

11.4.1. Les méthodes de la classe MenuBar

Méthodes	Role
void add(Menu)	ajouter un menu dans la barre

int countMenus()	renvoie le nombre de menus  utiliser la méthode getMenuCount().
Menu getMenu(int pos)	renvoie le menu à la position spécifiée
void remove(int pos)	supprimer le menu à la position spécifiée
void remove(Menu)	supprimer le menu de la barre de menu

11.4.2. Les méthodes de la classe Menu

Méthodes	Role
MenuItem add(MenuItem) void add(String)	ajouter une option dans le menu
void addSeparator()	ajouter un trait de séparation dans le menu
int countItems()	renvoie le nombre d'options du menu  utiliser la méthode getItemCount().
MenuItem getItem(int pos)	déterminer l'option du menu à la position spécifiée
void remove(MenuItem mi)	supprimer la commande spécifiée
void remove(int pos)	supprimer la commande à la position spécifiée

11.4.3. Les méthodes de la classe MenuItem

Méthodes	Role
void disable()	désactiver l'élément  utiliser la méthode setEnabled(false).
void enable()	activer l'élément  utiliser la méthode setEnabled(true).
void enable(boolean cond)	désactiver ou activer l'élément en fonction du paramètre  utiliser la méthode setEnabled(boolean).
String getLabel()	Renvoie le texte de l'élément
boolean isEnabled()	renvoie l'état de l'élément (actif / inactif)
void setLabel(String text)	définir une nouveau texte pour la commande

11.4.4. Les méthodes de la classe CheckboxMenuItem

Méthodes	Role
boolean getState()	renvoie l'état d'activation de l'élément

Void setState(boolean)	définir l'état d'activation de l'élément

12. La création d'interfaces graphiques avec AWT

12.1. Le dimensionnement des composants

En principe, il est automatique grace au LayoutManager. Pour donner à un composant une taille donnée, il faut redéfinir la méthode getPreferedSize de la classe Component.

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MonBouton extends Button {

   public Dimension getPreferredSize() {
      return new Dimension(800, 250);
   }

}
```

Le méthode getPreferedSize() indique la taille souhaitée mais pas celle imposée. En fonction du Layout Manager, le composant pourra ou non imposer sa taille.

Layout	Hauteur	Largeur
Sans Layout	oui	oui
FlowLayout	oui	oui
BorderLayout(East, West)	non	oui
BorderLayout(North, South)	oui	non
BorderLayout(Center)	non	non
GridLayout	non	non

Cette méthode oblige à sous classer tous les composants.

Une autre façon de faire est de se passer des Layout et de placer les composants à la main en indiquant leurs coordonnées et leurs dimensions.

Pour supprimer le Layout par défaut d'une classe, il faut appeler la méthode setLayout() avec comme paramètre null.

Trois méthodes de la classe Component permettent de positionner des composants :

- setBounds(int x, int y, int largeur, int hauteur)
- setLocation(int x , int y)
- setSize(int largeur, int hauteur)

Ces méthodes permettent de placer un composant à la position (x,y) par rapport au conteneur dans lequel il est inclus et d'indiquer sa largeur et sa hauteur.

Toutefois, les Layout Manager constituent un des facteurs importants de la portabilité des interfaces graphiques notamment en gérant la disposition et le placement des composants après redimentionnement du conteneur.

12.2. Le positionnement des composants

Lorsqu'on intègre un composant graphique dans un conteneur, il n'est pas nécessaire de préciser son emplacement car il est déterminé de façon automatique : la mise en forme est dynamique. On peut influencer cette mise en page en utilisant un gestionnaire de mise en page (Layout Manager) qui définit la position de chaque composant inséré. Dans ce cas, la position spécifiée est relative par rapport aux autres composants.

Chaque layout manager implémente l'interface java.awt.LayoutManager.

Il est possible d'utiliser plusieurs gestionnaires de mise en forme pour définir la présentation des composants. Par défaut, c'est la classe FlowLayout qui est utilisée pour la classe Panel et la classe BorderLayout pour Frame et Dialog.

Pour affecter une nouvelle mise en page, il faut utiliser la méthode setLayout() de la classe Container.

Exemple (code Java 1.1) :

```
  Panel p = new Panel();
   FlowLayout fl = new GridLayout(5,5);
   p.setLayout(fl);

   // ou p.setLayout( new GridLayout(5,5));
```

Les layout manager ont 3 avantages :

- l'aménagement des composants graphiques est délégué aux layout manager (il est inutile d'utiliser les coordonnées absolues)
- en cas de redimensionnement de la fenêtre, les contrôles sont automatiquement agrandis ou réduits
- ils permettent une indépendance vis à vis des plateformes.

Pour créer un espace entre les composants et le bord de leur conteneur, il faut rédifinir la méthode getInsets() d'un conteneur : cette méthode est héritée de la classe Container.

Exemple (code Java 1.1) :

```
public Insets getInsets() {
   Insets normal = super.getInsets();
   return new Insets(normal.top + 10, normal.left + 10,
   normal.bottom + 10, normal.right + 10);
}
```

Cet exemple permet de laisser 10 pixels en plus entre chaque bords du conteneur.

12.2.1. La mise en page par flot (FlowLayout)

La classe FlowLayout (mise en page flot) place les composants ligne par ligne de gauche à droite. Chaque ligne est complétée progressivement jusqu'à être remplie, puis passe à la suivante. Chaque ligne est centrée par défaut. C'est la mise en page par défaut des applets.

Il existe plusieurs constructeurs :

Constructeur	Role
FlowLayout( );	
FlowLayout(int align);	Permet de préciser l'alignement des composants dans le conteneur (CENTER, LEFT, RIGHT ...). Par défaut, align vaut CENTER
FlowLayout(int, int hgap, int vgap);	Permet de préciser et l'alignement horizontal et vertival dont la valeur par défaut est 5.

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
      setLayout(new FlowLayout());
      add(new Button("Bouton 1"));
      add(new Button("Bouton 2"));
      add(new Button("Bouton 3"));

      pack();

      show(); // affiche la fenetre

   }

   public static void main(String[] args) {

      new MaFrame();

   }

}
```

Chaque applet possède une mise en page flot implicitement initialisée à FlowLayout(FloawLayout.CENTER,5,5).

FlowLayout utilise les dimensions de son conteneur comme seul prinicipe de mise en forme des composants. Si les dimensions du conteneurs changent, le positionnement des composants est recalculé.

Exemple : la fenêtre précédente est simplement redimensionnée

12.2.2. La mise en page bordure (BorderLayout)

Avec ce Layout Manager, la disposition des composants est commandée par une mise en page en bordure qui découpe la surface en cinq zones : North, South, East, West, Center. On peut librement utiliser une ou plusieurs zones.

BorderLayout consacre tout l'espace du conteneur aux composants. Le composant du milieu dispose de la place inutilisée par les autres composants.

Il existe plusieurrs constructeurs :

Constructeur	Role
BorderLayout( )	
BorderLayout (int hgap,int vgap)	Permet de préciser l'espacement horizontal et vertical des composants.

Exemple (code Java 1.1) :

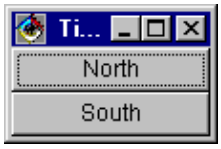
```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle("
      Titre de la Fenetre ");
      setSize(300, 150);
      setLayout(new
      BorderLayout());
      add("North", new Button(" bouton haut "));
      add("South", new Button(" bouton bas "));
      add("West",  new Button(" bouton gauche "));
      add("East",  new Button(" bouton droite "));
      add("Center", new Button(" bouton milieu "));
      pack();
      show(); // affiche la fenetre
   }

   public static void
      main(String[] args) {
      new MaFrame();
   }
}
```

Il est possible d'utiliser deux méthodes add surchargées de la classe Container : add(String, Component) ou le premier paramètre précise l'orientation du composants ou add(Component, Objet) ou le second paramètre précise la position sous forme de constante définie dans la classe BorderLayout.

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
```

```
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
      setLayout(new BorderLayout());
      add(new Button("North"), BorderLayout.NORTH);
      add(new Button("South"), BorderLayout.SOUTH);
      pack();
      show(); // affiche la fenetre

   }

   public static void main(String[] args) {

      new MaFrame();

   }

}
```



12.2.3. La mise en page de type carte (CardLayout)

Ce layout manager aide à construire des boîtes de dialogue composées de plusieurs onglets. Un onglet se compose généralement de plusieurs contrôles : on insère des panneaux dans la fenêtre utilisée par le CardLayout Manager. Chaque panneau correspond à un onglet de boîte de dialogue et contient plusieurs contrôles. Par défaut, c'est le premier onglet qui est affiché.

Ce layout possède deux constructeurs :

Constructeurs	Role
CardLayout()	
CardLayout(int, int)	Permet de préciser l'espace horizontal et vertical du tour du composant

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {

      super();
      setTitle("Titre de la Fenetre ");
      setSize(300,150);
      CardLayout cl = new CardLayout();
      setLayout(cl);

      //création d'un panneau contenant les contrôles d'un onglet
      Panel p = new Panel();

      //ajouter les composants au panel
      p.add(new Button("Bouton 1 panneau 1"));
      p.add(new Button("Bouton 2 panneau 1"));

      //inclure le panneau dans la fentre sous le nom "Page1"
      // ce nom est utilisé par show()

      add("Page1",p);
```

```
      //déclaration et insertion de l'onglet suivant
      p = new Panel();
      p.add(new Button("Bouton 1 panneau 2"));
      add("Page2", p);

      // affiche la fenetre
      pack();
      show();

   }

   public static void main(String[] args) {
      new MaFrame();
   }

}
```

Lors de l'insertion d'un onglet, un nom doit lui être attribué. Les fonctions nécessaires pour afficher un onglet de boîte de dialogue ne sont pas fournies par les méthodes du conteneur, mais seulement par le Layout Manager. Il est nécessaire de sauvegarder temporairement le Layout Manager dans une variable ou déterminer le gestionnaire en cours par un appel à getLayout(). Pour appeler un onglet donné, il faut utiliser la méthode show() du CardLayout Manager.

Exemple (code Java 1.1) :

```
((CardLayout)getLayout()).show(this, "Page2");
```

Les méthodes first(), last(), next() et previous() servent à parcourir les onglets de boîte de dialogue :

Exemple (code Java 1.1) :

```
((CardLayout)getLayout()).first(this);
```

12.2.4. La mise en page GridLayout

Ce Layout Manager établit un réseau de cellules identiques qui forment une sorte de quadrillage invisible : les composants sont organisés en lignes et en colonnes. Les éléments insérés dans la grille ont tous la même taille. Les cellules du quadrillage se remplissent de droite à gauche ou de haut en bas.

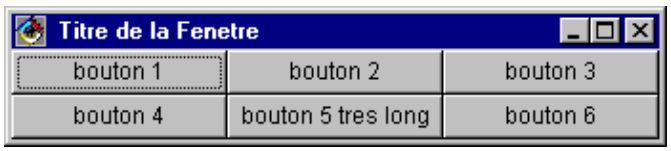
Il existe plusieurs constructeurs :

Constructeur	Role
GridLayout(int, int);	Les deux premiers entiers spécifient le nombre de lignes ou de colonnes de la grille.
GridLayout(int, int, int, int);	permet de préciser en plus l'espacement horizontal et vertical des composants.

Exemple (code Java 1.1) :

```
import java.awt.*;
```

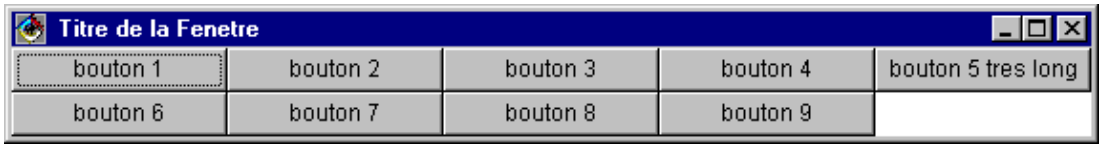
```
public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
      setLayout(new GridLayout(2, 3));
      add(new Button("bouton 1"));
      add(new Button("bouton 2"));
      add(new Button("bouton 3"));
      add(new Button("bouton 4"));
      add(new Button("bouton 5 tres long"));
      add(new Button("bouton 6"));

      pack();

      show(); // affiche la fenetre

   }

   public static void main(String[] args) {
      new MaFrame();
   }

}
```



Attention : lorsque le nombre de ligne et de colonne est spécifié alors le nombre de colonne est ignoré. Ainsi par Exemple GridLayout(5,4) est équivalent à GridLayout(5,0).

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
      setLayout(new GridLayout(2, 3));
      add(new Button("bouton 1"));
      add(new Button("bouton 2"));
      add(new Button("bouton 3"));
      add(new Button("bouton 4"));
      add(new Button("bouton 5 tres long"));
      add(new Button("bouton 6"));
      add(new Button("bouton 7"));
      add(new Button("bouton 8"));
      add(new Button("bouton 9"));

      pack();
      show(); // affiche la fenetre

   }

   public static void
      main(String[] args) {
      new MaFrame();
   }

}
```



12.2.5. La mise en page GridBagLayout

Ce gestionnaire (grille étendue) est le plus riche en fonctionnalités : le conteneur est divisé en cellules égales mais un composant peut occuper plusieurs cellules de la grille et il est possible de faire une distribution dans des cellules distinctes. Un objet de la classe GridBagConstraints permet de donner les indications de positionnement et de dimension à l'objet GridBagLayout.

Les lignes et les colonnes prennent naissance au moment ou les contrôles sont ajoutés. Chaque contrôle est associé à un objet de la classe GridBagConstraints qui indique l'emplacement voulu pour le contrôle.

Exemple (code Java 1.1) :

```
GridBagLayout gb1 = new GridBagLayout( );
GridBagConstraints gbc = new GridBagConstraints( );
```

Les variables d'instances pour manipuler l'objet GridBagLayoutConstrainst sont :

Variable	Role
gridx et gridy	Ces variables contiennent les coordonnées de l'origine de la grille. Elles permettent un positionnement précis à une certaine position d'un composant. Par défaut elles ont la valeur GrigBagConstraint.RELATIVE qui indique qu'un composant se range à droite du précédent
gridwidth, gridheight	Définissent combien de cellules va occuper le composant (en hauteur et largeur). Par défaut la valeur est 1. L'indication est relative aux autres composants de la ligne ou de la colonne. La valeur GridBagConstraints.REMAINDER spécifie que le prochain composant inséré sera le dernier de la ligne ou de la colonne courante. La valeur GridBagConstraints.RELATIVE place le composant après le dernier composant d'une ligne ou d'une colonne.
fill	Définit le sort d'un composant plus petit que la cellule de la grille. GridBagConstraints.NONE conserve la taille d'origine : valeur par défaut GridBagConstraints.HORIZONTAL dilaté horizontalement GridBagConstraints.VERTICAL dilaté verticalement GridBagConstraints.BOTH dilatés aux dimensions de la cellule
ipadx, ipady	Permettent de définir l'agrandissement horizontal et vertical des composants. Ne fonctionne que si une dilatation est demandée par fill. La valeur par défaut est (0,0).
anchor	Lorsqu'un composant est plus petit que la cellule dans laquelle il est inséré, il peut être positionné à l'aide de cette variable pour définir le côté par lequel le contrôle doit être aligné dans la cellule. Les variables possibles sont NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST et EAST
weightx, weighty	Permettent de définir la répartition de l'espace en cas de changement de dimension

Exemple (code Java 1.1) :

```
import java.awt.*;

public class MaFrame extends Frame {

   public MaFrame() {
      super();
      setTitle(" Titre de la Fenetre ");
      setSize(300, 150);
```

```
      Button b1 = new Button(" bouton 1 ");
      Button b2 = new Button(" bouton 2 ");
      Button b3 = new Button(" bouton 3 ");

      GridBagLayout gb = new GridBagLayout();

      GridBagConstraints gbc = new GridBagConstraints();
      setLayout(gb);

      gbc.fill = GridBagConstraints.BOTH;
      gbc.weightx = 1;
      gbc.weighty = 1;
      gb.setConstraints(b1, gbc); // mise en forme des objets
      gb.setConstraints(b2, gbc);
      gb.setConstraints(b3, gbc);

      add(b1);
      add(b2);
      add(b3);

      pack();

      show(); // affiche la fenetre

   }

   public static void main(String[] args) {
      new MaFrame();
   }

}
```

Cet exemple place trois boutons l'un à coté de l'autre. Ceci permet en cas de changement de dimension du conteneur de conserver la mise en page : la taille des composants est automatiquement ajustée.

Pour placer les 3 boutons l'un au dessus de l'autre, il faut affecter la valeur 1 à la variable gbc.gridx.

12.3. La création de nouveaux composants à partir de Panel

Il est possible de définir de nouveau composant qui hérite directement de Panel

Exemple (code Java 1.1) :

```
class PanneauClavier extends Panel {

   PanneauClavier()
   {
      setLayout(new GridLayout(4,3));

      for (int num=1; num <= 9 ; num++) add(new Button(Integer.toString(num)));
      add(new Button(«*»);
      add(new Button(«0»);
      add(new Button(«# »);
```

```
   }
}
public class demo exteds Applet {
   public void init() { add(new PanneauClavier()); }
}
```

12.4. Activer ou desactiver des composants

L'activation ou la désactivation d'un composant se fait grace à sa méthode setEnabled(boolean). La valeur booléenne passée en paramètres indique l'état du composant (false : interdit l'usage du composant). Cette méthode est un moyen d'interdire à un composant d'envoyer des évenements utilisateurs.

12.5. Afficher une image dans une application.

L'image doit préalablement être chargée grâce à la classe Toolkit.

La suite de ce chapitre sera développée dans une version future de ce document

13. L'interception des actions de l'utilisateur

N'importe quelle interface graphique doit interagir avec l'utilisateur et donc réagir a certains événements. Le modèle de gestion de ces évenements à changer entre le JDK 1.0 et 1.1.

Ce chapitre traite de la capture des ces événements pour leur associer des traitements. Il contient plusieurs sections :

13.1. Intercepter les actions de l'utilisateur avec Java version 1.0

Cette section sera développée dans une version future de ce document

13.2. Intercepter les actions de l'utilisateur avec Java version 1.1

Les événements utilisateurs sont gérés par plusieurs interfaces EventListener.

Les interfaces EventListener permettent à un composant de générer des événements utilisateurs. Une classe doit contenir une interface auditeur pour chaque type de composant :

- ActionListener : clic de souris ou enfoncement de la touche Enter
- ItemListener : utilisation d'une liste ou d'une case à cocher
- MouseMotionListener : evénément de souris
- WindowListener : événement de fenêtre

L'ajout d'une interface EventListener impose plusieurs ajouts dans le code :

1. importer le groupe de classe java.awt.event

Exemple (code Java 1.1) :
`import java.awt.event.*;`

2. la classe doit déclarer qu'elle utilisera une ou plusieurs interfaces d'écoute

Exemple (code Java 1.1) :

```
public class AppletAction extends Applet implements ActionListener{
```

Pour déclarer plusieurs interfaces, il suffit de les séparer par des virgules

Exemple (code Java 1.1) :

```
public class MonApplet extends Applet implements ActionListener, MouseListener {
```

3. Appel à la méthode addXXX() pour enregistrer l'objet qui gerera les évenements XXX du composant

   Il faut configurer le composant pour qu'il possède un "écouteur" pour l'événement utilisateur concerné.

Exemple (code Java 1.1) : création d'un bouton capable de réagir à un evénements

```
Button b = new Button(«boutton»);
b.addActionListener(this);
```

Ce code crée l'objet de la classe Button et appelle sa méthode addActionListener(). Cette méthode permet de préciser qu'elle sera la classe qui va gérer l'évenement utilisateur de type ActionListener du bouton. Cette classe doit impérativement implémenter l'interface de type EventListener correspondante soit dans cette exemple ActionListener. L'instruction this indique que la classe elle même recevra et gérera l'évenement utilisateur.

L'apparition d'un évenement utilisateur généré par un composant doté d'un auditeur appelle automatiquement une méthode, qui doit se trouver dans la classe référencée dans l'instruction qui lie l'auditeur au composant. Dans l'exemple, cette méthode doit être située dans la même classe parce que c'est l'objet lui même qui est spécifié avec l'instruction this. Une autre classe indépendante peut être utilisée : dans ce cas il faut préciser une instance de cette classe en temps que paramètre.

4. implémenter les méthodes déclarées dans les interfaces

   Chaque auditeur possède des méthodes différentes qui sont appelées pour traiter leurs évenéments. Par exemple, l'interface ActionListener envoie des évenements à une classe nommée actionPerformed( ).

Exemple (code Java 1.1) :

```
public void actionPerformed(ActionEvent evt) {
   //insérer ici le code de la méthode
};
```

Pour identifier le composant qui a généré l'evénement il faut utiliser la méthode getActionCommand() de l'objet ActionEvent fourni en paramètre de la méthode :

Exemple (code Java 1.1) :

```
String composant = evt.getActionCommand();
```

getActionCommand renvoie une chaîne de caractères. Si le composant est un bouton, alors il renvoie le texte du bouton, si le composant est une zone de saisie, c'est le texte saisie qui sera renvoyé (il faut appuyer sur "Entrer" pour générer l'événement), etc ...

La méthode getSource() renvoie l'objet qui a généré l'événement. Cette méthode est plus sure que la précédente

Exemple (code Java 1.1) :

```
Button b = new Button(« bouton »);

...
```

```
void public actionPerformed(actionEvent evt) {
   Object source = evt.getSource();

   if (source == b) // action a effectuer
}
```

La méthode getSource() peut être utilisé avec tous les évenements utilisateur.

Exemple (code Java 1.1) : Exemple complet qui affiche le composant qui a généré l'événement

```
package applets;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class AppletAction extends Applet implements ActionListener{

   public void actionPerformed(ActionEvent evt) {
      String composant = evt.getActionCommand();
      showStatus("Action sur le composant : " + composant);
   }


   public void init() {
      super.init();

      Button b1 = new Button("boutton 1");
      b1.addActionListener(this);
      add(b1);

      Button b2 = new Button("boutton 2");
      b2.addActionListener(this);
      add(b2);

      Button b3 = new Button("boutton 3");
      b3.addActionListener(this);
      add(b3);
   }
}
```

13.2.1. L'interface ItemListener

Cette interface permet de réagir à la sélection de cases à cocher et de liste d'options. Pour qu'un composant genère des évenements, il faut utiliser la méthode addItemListener().

Exemple (code Java 1.1) :

```
Checkbox cb = new Checkbox(« choix »,true);
cb.addItemListener(this);
```

Ces évenements sont reçus par la méthode itemStateChanged() qui attend un objet de type ItemEvent en argument

Pour déterminer si une case à cocher est sélectionnée ou inactive, utiliser la méthode getStateChange() avec les constantes ItemEvent.SELECTED ou ItemEvent.DESELECTED.

Exemple (code Java 1.1) :

```
package applets;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class AppletItem extends Applet implements ItemListener{

   public void init() {
      super.init();
      Checkbox cb = new Checkbox("choix 1", true);
      cb.addItemListener(this);
      add(cb);
   }

   public void itemStateChanged(ItemEvent item) {
      int status = item.getStateChange();
      if (status == ItemEvent.SELECTED)
         showStatus("choix selectionne");
      else
         showStatus("choix non selectionne");
   }

}
```

Pour connaitre l'objet qui a généré l'événement, il faut utiliser la méthode getItem().

Pour déterminer la valeur sélectionnée dans une combo box, il faut utiliser la méthode getItem() et convertir la valeur en chaîne de caractères.

Exemple (code Java 1.1) :

```
package applets;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class AppletItem extends Applet implements ItemListener{

   public void init() {
      Choice c = new Choice();
      c.add("choix 1");
      c.add("choix 2");
      c.add("choix 3");
      c.addItemListener(this);
      add(c);
}

   public void itemStateChanged(ItemEvent item) {
      Object obj = item.getItem();
      String selection = (String)obj;
      showStatus("choix : "+selection);
   }
}
```

13.2.2. L'interface TextListener

Cette interface permet de réagir aux modifications de la zone de saisie ou du texte.

La méthode addTextListener() permet à un composant de texte de générer des événements utilisateur. La méthode TextValueChanged() reçoit les évenements.

Exemple (code Java 1.1) :

```
package applets;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class AppletText extends Applet implements TextListener{

```

```
   public void init() {
      super.init();

      TextField t = new TextField("");
      t.addTextListener(this);
      add(t);
   }

   public void textValueChanged(TextEvent txt) {
      Object source = txt.getSource();
      showStatus("saisi = "+((TextField)source).getText());
   }
}
```

13.2.3. L'interface MouseMotionListener

La méthode addMouseMotionListener() permet de gérer les évenements liés à des mouvements de souris. La méthode mouseDragged() et mouseMoved() reçoivent les évenements.

Exemple (code Java 1.1) :

```
package applets;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class AppletMotion extends Applet implements MouseMotionListener{
   private int x;
   private int y;

   public void init() {
      super.init();
      this.addMouseMotionListener(this);
   }

   public void mouseDragged(java.awt.event.MouseEvent e) {}

   public void mouseMoved(MouseEvent e) {
      x = e.getX();
      y = e.getY();
      repaint();
      showStatus("x = "+x+" ; y = "+y);
   }

   public void paint(Graphics g) {
      super.paint(g);
      g.drawString("x = "+x+" ; y = "+y,20,20);
   }
}
```

13.2.4. L'interface MouseListener

Cette interface permet de réagir aux clics de souris. Les méthodes de cette interface sont :

- public void mouseClicked(MouseEvent e);
- public void mousePressed(MouseEvent e);
- public void mouseReleased(MouseEvent e);
- public void mouseEntered(MouseEvent e);
- public void mouseExited(MouseEvent e);

Exemple (code Java 1.1) :

```
package applets;

import java.applet.*;
```

```
import java.awt.*;
import java.awt.event.*;

public class AppletMouse extends Applet implements MouseListener {
   int nbClick = 0;

   public void init() {
      super.init();
      addMouseListener(this);
   }

   public void mouseClicked(MouseEvent e) {
      nbClick++;
      repaint();
   }

   public void mouseEntered(MouseEvent e) {}

   public void mouseExited(MouseEvent e) {}

   public void mousePressed(MouseEvent e) {}

   public void mouseReleased(MouseEvent e) {}

   public void paint(Graphics g) {
      super.paint(g);
      g.drawString("Nombre de clics : "+nbClick,10,10);
   }
}
```

Une classe qui implémente cette interface doit définir ces 5 méthodes. Si toutes les méthodes ne doivent pas être utilisées, il est possible de définir une classe qui hérite de MouseAdapter. Cette classe fournit une implémentation par défaut de l'interface MouseListener.

Exemple (code Java 1.1) :

```
class gestionClics extends MouseAdapter {

   public void mousePressed(MouseEvent e) {
      //traitement
   }
}
```

Dans le cas d'une classe qui hérite d'une classe Adapter, il suffit de redéfinir la ou les méthodes qui contiendront du code pour traiter les événements concernés. Par défaut, les différentes méthodes définies dans l'Adapter ne font rien.

Cette nouvelle classe ainsi définie doit être passée en paramètre à la méthode addMouseListener() au lieu de this qui indiquait que la classe répondait elle même à l'événement.

13.2.5. L'interface WindowListener

La méthode addWindwowListener() permet à un objet Frame de générer des événements. Les méthodes de cette interface sont :

- public void windowOpened(WindowEvent e)
- public void windowClosing(WindowEvent e)
- public void windowClosed(WindowEvent e)
- public void windowIconified(WindowEvent e)
- public void windowDeinconified(WindowEvent e)
- public void windowActivated(WindowEvent e)
- public void windowDeactivated(WindowEvent e)

windowClosing est appelée lorque l'on clique sur la case système de fermeture de la fenêtre. windowClosed est appelé après la fermeture de la fenêtre : cette méthode n'est utile que si la fermeture de la fenêtre n'entraine pas la fin de l'application.

Exemple (code Java 1.1) :

```
package test;

import java.awt.event.*;

class GestionnaireFenetre extends WindowAdpter {

   public void windowClosing(WindowEvent e) {
      System.exit(0);
   }
}
```

Exemple (code Java 1.1) :

```
package test;

import java.awt.*;
import java.awt.event.*;

public class TestFrame extends Frame {

   private GestionnaireFenetre gf = new GestionnaireFenetre();


   public TestFrame(String title) {
     super(title);
     addWindowListener(gf);
   }

   public static void main(java.lang.String[] args) {
      try {
         TestFrame tf = new TestFrame("TestFrame");
         tf.setVisible(true);
      } catch (Throwable e) {
         System.err.println("Erreur");
         e.printStackTrace(System.out);
      }
   }
}
```

13.2.6. Les différentes implémentations des Listener

La mise en oeuvre des Listeners peut se faire selon différentes formes : la classe implémentant elle même l'interface, une classe indépendante, une classe interne, une classe interne anonyme.

13.2.6.1. Une classe implémentant elle même le listener

Exemple (code Java 1.1) :

```
package test;

import java.awt.*;
import java.awt.event.*;

public class TestFrame3 extends Frame implements WindowListener {

   public TestFrame3(String title) {
      super(title);
      this.addWindowListener(this);
   }
```

```
   public static void main(java.lang.String[] args) {
      try {
         TestFrame3 tf = new TestFrame3("testFrame3");
         tf.setVisible(true);
      } catch (Throwable e) {
         System.err.println("Erreur");
         e.printStackTrace(System.out);
      }
   }

   public void windowActivated(java.awt.event.WindowEvent e) {}

   public void windowClosed(java.awt.event.WindowEvent e) {}

   public void windowClosing(java.awt.event.WindowEvent e) {
      System.exit(0);
   }

   public void windowDeactivated(java.awt.event.WindowEvent e) {}

   public void windowDeiconified(java.awt.event.WindowEvent e) {}

   public void windowIconified(java.awt.event.WindowEvent e) {}

   public void windowOpened(java.awt.event.WindowEvent e) {}

}
```

13.2.6.2. Une classe indépendante implémentant le listener

Exemple (code Java 1.1) :

```
package test;

import java.awt.*;
import java.awt.event.*;

public class TestFrame4 extends Frame {

   public TestFrame4(String title) {
      super(title);
      gestEvt ge = new gestEvt();
      addWindowListener(ge);
   }

   public static void main(java.lang.String[] args) {
      try {
         TestFrame4 tf = new TestFrame4("testFrame4");
         tf.setVisible(true);
      } catch (Throwable e) {
         System.err.println("Erreur");
         e.printStackTrace(System.out);
      }
   }
}
```

Exemple (code Java 1.1) :

```
package test;

import java.awt.event.*;

public class gestEvt implements WindowListener {

   public void windowActivated(WindowEvent e) {}
   public void windowClosed(WindowEvent e) {}
   public void windowClosing(WindowEvent e) {
      System.exit(0);
   }
   public void windowDeactivated(WindowEvent e) {}
```

```
   public void windowDeiconified(WindowEvent e) {}
   public void windowIconified(WindowEvent e) {}
   public void windowOpened(WindowEvent e) {}
}
```

13.2.6.3. Une classe interne

Exemple (code Java 1.1) :

```
package test;

import java.awt.*;
import java.awt.event.*;

public class TestFrame2 extends Frame {

   class gestEvt implements WindowListener {
      public void windowActivated(WindowEvent e) {};
      public void windowClosed(WindowEvent e) {};
      public void windowClosing(WindowEvent e) {
         System.exit(0);
      };
      public void windowDeactivated(WindowEvent e) {};
      public void windowDeiconified(WindowEvent e) {};
      public void windowIconified(WindowEvent e) {};
      public void windowOpened(WindowEvent e) {};
   };

   private gestEvt ge = new TestFrame2.gestEvt();


   public TestFrame2(String title) {
     super(title);
     addWindowListener(ge);
   }

   public static void main(java.lang.String[] args) {
      try {
         TestFrame2 tf = new TestFrame2("TestFrame2");
         tf.setVisible(true);
      } catch (Throwable e) {
         System.err.println("Erreur");
         e.printStackTrace(System.out);
      }
   }
}
```

13.2.6.4. Une classe interne anonyme

Exemple (code Java 1.1) :

```
package test;

import java.awt.*;
import java.awt.event.*;

public class TestFrame1 extends Frame {

   public TestFrame1(String title) {
      super(title);
      addWindowListener(new WindowAdapter() {
         public void windowClosed(.WindowEvent e) {
            System.exit(0);
         };
      });
   }

   public static void main(java.lang.String[] args) {
      try {
```

```
         TestFrame1 tf = new TestFrame1("TestFrame");
         tf.setVisible(true);
      } catch (Throwable e) {
         System.err.println("Erreur");
         e.printStackTrace(System.out);
      }
   }
}
```

13.2.7. Résumé

Le mécanisme mis en place pour intercepter des événements est le même quel que soit ces événements :

- associer au composant qui est à l'origine de l'évenement un contrôleur adéquat : utilisation des méthodes addXXXListener() Le paramètre de ces méthodes indique l'objet qui a la charge de répondre au message : cet objet doit implémenter l'interface XXXListener correspondant ou dérivé d'une classe XXXAdapter (créer une classe qui implémente l'interface associé à l'événement que l'on veut gérer. Cette classe peut être celle du composant qui est à l'origine de l'événement (facilité d'implémentation) ou une classe indépendante qui détermine la frontière entre l'interface graphique (émission d'événement) et celle qui représente la logique de l'application (traitement des événements)).
- les classes XXXAdapter sont utiles pour créer des classes dédiées au traitement des évenements car elles implémentent des méthodes par défaut pour celles définies dans l'interface XXXListener dérivées de EventListener. Il n'existe une classe Adapter que pour les interface qui possédent plusieurs méthodes.
- implémenter la méthode associé à l'événement qui fournit en paramètre un objet de type AWTEvent (classe mère de tout événement) qui contient des informations utiles (position du curseur, état du clavier ...).

14. Le développement d'interfaces graphiques avec SWING

Swing fait partie de la bibliothèque Java Foundation Classes (JFC). C'est une API dont le but est similaire à celui de l'API AWT mais dont le mode de fonctionnement et d'utilisation est conplètement différent. Swing a été intégrée au JDK depuis sa version 1.2. Cette bibliothèque existe séparement pour le JDK 1.1.

La bibliothèque JFC contient :

- l'API Swing : de nouvelles classes et interfaces pour construire des interfaces graphiques
- Accessibility API :
- 2D API: support du graphisme en 2D
- API pour l'impression et le cliquer/glisser

Ce chapitre contient plusieurs sections :

14.1. Présentation de Swing

Swing propose de nombreux composants dont certains possèdent des fonctions étendues, une utilisation des mécanismes de gestion d'événements performants (ceux introduits par le JDK 1.1) et une apparence modifiable à la volée (une interface graphique qui emploie le style du système d'exploitation Windows ou Motif ou un nouveau style spécifique à Java nommé Metal).

Tous les éléments de Swing font partie d'un package qui a changé plusieurs fois de nom : le nom du package dépend de la version du J.D.K. utilisée :

- com.sun.java.swing : jusqu'à la version 1.1 beta 2 de Swing, de la version 1.1 des JFC et de la version 1.2 beta 4 du J.D.K.
- java.awt.swing : utilisé par le J.D.K. 1.2 beta 2 et 3
- javax.swing : à partir des versions de Swing 1.1 beta 3 et J.D.K. 1.2 RC1

Les composants Swing forment un nouvelle hériarchie parallèle à celle de l'AWT. L'ancêtre de cette hiérarchie est le composant JComponent. Presque tous ces composants sont écrits en pur Java : ils ne possèdent aucune partie native sauf ceux qui assurent l'interface avec le système d'exploitation : JApplet, JDialog, JFrame, et JWindow. Cela permet aux composants de toujours avoir la même apparence quelque soit le système sur lequel l'application s'exécute.

Tous les composants Swing possèdent les caractéristiques suivantes :

- ce sont des beans
- ce sont des composants légers (pas de partie native) hormis quelques exceptions.
- leurs bords peuvent être changés

La procédure à suivre pour utiliser un composant Swing est identique à celle des composants de la bibliothèque AWT : créer le composant en appelant son constructeur, appeler les méthodes du composant si nécessaire pour le personnaliser et l'ajouter dans un conteneur.

Swing utilise la même infrastructure de classes que AWT, ce qui permet de mélanger des composants Swing et AWT dans la même interface. Sun recommande toutefois d'éviter de les mélanger car certains peuvent ne pas être restitués correctement.

Les composants Swing utilisent des modèles pour contenir leurs états ou leur données. Ces modèles sont des classes particulières qui possèdent toutes un comportement par défaut.

14.2. Les packages Swing

Swing contient plusieurs packages :

javax.swing	package principal : il contient les interfaces, les principaux composants, les modèles par défaut
javax.swing.border	Classes représentant les bordures
javax.swing.colorchooser	Classes définissant un composant pour la sélection de couleurs
javax.swing.event	Classes et interfaces pour les événements spécifiques à Swing. Les autres événements sont ceux de AWT (java.awt.event)
javax.swing.filechooser	Classes définissant un composant pour la sélection de fichiers
javax.swing.plaf	Classes et interfaces génériques pour gérer l'apparence
javax.swing.plaf.basic	Classes et interfaces de base pour gérer l'apparence
javax.swing.plaf.metal	Classes et interfaces pour définir l'apparence Metal qui est l'apparence par défaut
javax.swing.table	Classes définissant un composant pour la présentation de données sous forme de tableau
javax.swing.text	Classes et interfaces de bases pour les composants manipulant du texte
javax.swing.text.html	Classes permettant le support du format HTML
javax.swing.text.html.parser	Classes permettant d'analyser des données au format HTML
javax.swing.text.rtf	Classes permettant le support du format RTF
javax.swing.tree	Classes définissant un composant pour la présentation de données sous forme d'arbre
javax.swing.undo	Classes permettant d'implémenter les fonctions annuler/refaire

14.3. Un exemple de fenêtre autonome

La classe de base d'une application est la classe JFrame. Son rôle est équivalent à la classe Frame de l'AWT et elle s'utilise de la même façon.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.event.*;

public class swing1 extends JFrame {

   public swing1() {
      super("titre de l'application");

      WindowListener l = new WindowAdapter() {
         public void windowClosing(WindowEvent e){
            System.exit(0);
```

```
         }
      };

      addWindowListener(l);
      setSize(200,100);
      setVisible(true);
   }

   public static void main(String [] args){
      JFrame frame = new swing1();
   }
}
```

14.4. Les composants Swing

Il existe des composants Swing équivalents pour chacun des composants AWT avec des constructeurs semblables. De nombreux constructeurs acceptent comme argument un objet de type Icon, qui représente une petite image généralement stockée au format Gif.

Le constructeur d'un objet Icon admet comme seul paramètre le nom ou l'URL d'un fichier graphique

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.event.*;

public class swing3 extends JFrame {

  public swing3() {

     super("titre de l'application");

      WindowListener l = new WindowAdapter() {
         public void windowClosing(WindowEvent e){
            System.exit(0);
         }
      };

      addWindowListener(l);

      ImageIcon img = new ImageIcon("tips.gif");
      JButton bouton = new JButton("Mon bouton",img);

      JPanel panneau = new JPanel();
      panneau.add(bouton);
      setContentPane(panneau);
      setSize(200,100);
      setVisible(true);
   }

  public static void main(String [] args){
      JFrame frame = new swing3();
  }
}
```

14.4.1. La classe JFrame

JFrame est l'équivalent de la classe Frame de l'AWT : les principales différences sont l'utilisation du double buffering qui améliore les rafraichissements et l'utilisation d'un panneau de contenu (contentPane) pour insérer des composants (ils ne sont plus insérés directement au JFrame mais à l'objet contentPane qui lui est associé). Elle représente une fenêtre principale qui possède un titre, une taille modifiable et éventuellement un menu.

La classe possède plusieurs constructeurs :

Constructeur	Rôle

JFrame()	
JFrame(String)	Création d'une instance en précisant le titre

Par défaut, la fenêtre créée n'est pas visible. La méthode setVisible() permet de l'afficher.

Exemple (code Java 1.1) :

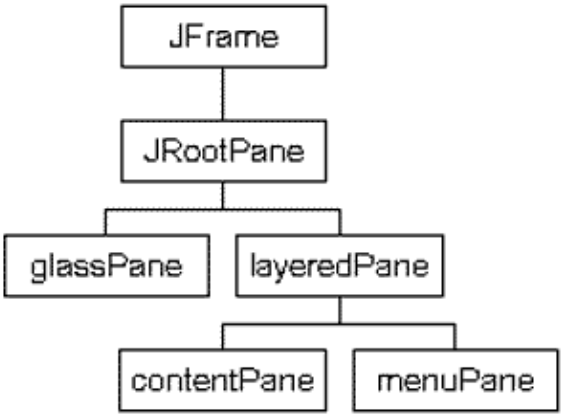
```
import javax.swing.*;

public class TestJFrame1 {

  public static void main(String argv[]) {
    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    f.setVisible(true);
  }
}
```

La gestion des événements est identique à celle utilisée dans l'AWT depuis le J.D.K. 1.1.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.event.*;

public class swing2 extends JFrame {

   public swing2() {

     super("titre de l'application");

      WindowListener l = new WindowAdapter() {
         public void windowClosing(WindowEvent e){
           System.exit(0);
         }
      };
     addWindowListener(l);

     JButton bouton = new JButton("Mon bouton");
     JPanel panneau = new JPanel();
     panneau.add(bouton);

     setContentPane(panneau);
     setSize(200,100);
     setVisible(true);
   }

   public static void main(String [] args){
      JFrame frame = new swing2();
   }
}
```

Tous les composants associés à un objet JFrame sont gérés par un objet de la classe JRootPane. Un objet JRootPane contient plusieurs Panes. Tous les composants ajoutés au JFame doivent être ajoutés à un des Pane du JRootPane et non au JFrame directement. C'est aussi à un de ces Panes qu'il faut associer un layout manager si nécessaire.



Le Pane le plus utilisé est le ContentPane. Le Layout manager par défaut du contentPane est BorderLayout. Il est possible de le changer :

Exemple (code Java 1.1) :

```
...
f.getContentPane().setLayout(new FlowLayout());
...
```

Exemple (code Java 1.1) :

```
import javax.swing.*;

public class TestJFrame2 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);
    f.setVisible(true);
  }
}
```

Le JRootPane se compose de plusieurs éléments :

- glassPane : un JPanel par défaut
- layeredPane qui se compose du contentPane (un JPanel par défaut) et du menuBar (un objet de type JMenuBar)

Le glassPane est un JPanel transparent qui se situe au dessus du layeredPane. Le glassPane peut être n'importe quel composant : pour le modifier il faut utiliser la méthode setGlassPane() en fournissant le composant en paramètre.

Le layeredPane regroupe le contentPane et le menuBar.

Le contentPane est par défaut un JPanel opaque dont le gestionnaire de présentation est un BorderLayout. Ce panel peut être remplacé par n'importe quel composant grâce à la méthode setContentPane().

Attention : il ne faut pas utiliser directement la méthode setLayout() d'un objet JFrame sinon une exception est levée.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.*;

public class TestJFrame7 {
```

```
  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setLayout(new FlowLayout());
    f.setSize(300,100);
    f.setVisible(true);
  }
}
```

Résultat :

```
C:\swing\code>java TestJFrame7
Exception in thread "main" java.lang.Error: Do not use javax.swing.JFrame.setLay
out() use javax.swing.JFrame.getContentPane().setLayout() instead
        at javax.swing.JFrame.createRootPaneException(Unknown Source)
        at javax.swing.JFrame.setLayout(Unknown Source)
        at TestJFrame7.main(TestJFrame7.java:8)
```

Le menuBar permet d'attacher un menu à la JFrame. Par défaut, le menuBar est vide. La méthode setJMenuBar() permet d'affecter un menu à la JFrame.

Exemple (code Java 1.1) : Création d'un menu très simple

```
import javax.swing.*;
import java.awt.*;

public class TestJFrame6 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    JMenuBar menuBar = new JMenuBar();
    f.setJMenuBar(menuBar);

    JMenu menu = new JMenu("Fichier");
    menu.add(menuItem);
    menuBar.add(menu);

    f.setVisible(true);
  }
}
```

14.4.1.1. Le comportement par défaut à la fermeture

Il est possible de préciser comment un objet JFrame, JInternalFrame, ou JDialog réagit à sa fermeture grâce à la méthode setDefaultCloseOperation(). Cette méthode attend en paramètre une valeur qui peut être :

Constante	Rôle
WindowConstants.DISPOSE_ON_CLOSE	détruit la fenêtre
WindowConstants.DO_NOTHING_ON_CLOSE	rend le bouton de fermeture inactif
WindowConstants.HIDE_ON_CLOSE	cache la fenêtre

Cette méthode ne permet pas d'associer d'autres traitements. Dans ce cas, il faut intercepter l'évenement et lui associer les traitements.

Exemple (code Java 1.1) : la fenêtre disparaît lors de sa fermeture mais l'application ne se termine pas.

```
import javax.swing.*;

public class TestJFrame3 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    f.setVisible(true);
  }
}
```

14.4.1.2. La personnalisation de l'icône

La méthode setIconImage() permet de modifier l'icône de la JFrame.

Exemple (code Java 1.1) :

```
import javax.swing.*;

public class TestJFrame4 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);
    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    ImageIcon image = new ImageIcon("book.gif");
    f.setIconImage(image.getImage());
    f.setVisible(true);

  }
}
```

Si l'image n'est pas trouvée, alors l'icône est vide. Si l'image est trop grande, elle est redimensionnée.

14.4.1.3. Centrer une JFrame à l'écran

Par défaut, une JFrame est affichée dans le coin supérieur gauche de l'écran. Pour la centrer dans l'écran, il faut procéder comme pour une Frame : déterminer la position de la Frame en fonction de sa dimension et de celle de l'écran et utiliser la méthode setLocation() pour affecter cette position.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.*;

public class TestJFrame5 {
```

```
  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    f.setLocation(dim.width/2 − f.getWidth()/2, dim.height/2 − f.getHeight()/2);

    f.setVisible(true);
  }
}
```

14.4.1.4. Les évenements associées à un JFrame

La gestion des évenements associés à un objet JFrame est identique à celle utilisée pour un objet de type Frame de AWT.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.event.*;

public class TestJFrame8 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
      }
    );

  }
}
```

14.4.2. Les étiquettes : la classe JLabel

Le composant JLabel propose les mêmes fonctionnalités que les intitulés AWT mais ils peuvent en plus contenir des icônes .

Cette classe possède plusieurs constructeurs :

Constructeurs	Rôle
JLabel()	Création d'une instance sans texte ni image
JLabel(Icon)	Création d'une instance en précisant l'image
JLabel(Icon, int)	Création d'une instance en précisant l'image et l'alignement horizontal
JLabel(String)	Création d'une instance en précisant le texte
JLabel(String, Icon, int)	Création d'une instance en précisant le texte, l'image et l'alignement horizontal
JLabel(String, int)	Création d'une instance en précisant le texte et l'alignement horizontal

Le composant JLabel permet d'afficher un texte et/ou une icône en précisant leur alignement. L'icône doit être au format GIF et peut être une animation dans ce format.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.*;

public class TestJLabel1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(100,200);

    JPanel pannel = new JPanel();
    JLabel jLabel1 =new JLabel("Mon texte dans JLabel");
    pannel.add(jLabel1);

    ImageIcon icone = new ImageIcon("book.gif");
    JLabel jLabel2 =new JLabel(icone);
    pannel.add(jLabel2);

    JLabel jLabel3 =new JLabel("Mon texte",icone,SwingConstants.LEFT);
    pannel.add(jLabel3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La classe JLabel définit plusieurs méthodes pour modifier l'apparence du composant :

Méthodes	Rôle
setText()	Permet d'initialiser ou de modifier le texte affiché
setOpaque()	Indique si le composant est transparent (paramètre false) ou opaque (true)
setBackground()	Indique la couleur de fond du composant (setOpaque doit être à true)
setFont()	Permet de préciser la police du texte
setForeGround()	Permet de préciser la couleur du texte
setHorizontalAlignment()	Permet de modifier l'alignement horizontal du texte et de l'icône
setVerticalAlignment()	Permet de modifier l'alignement vertical du texte et de l'icône
setHorizontalTextAlignment()	Permet de modifier l'alignement horizontal du texte uniquement
setVerticalTextAlignment()	Permet de modifier l'alignement vertical du texte uniquement Exemple : jLabel.setVerticalTextPosition(SwingConstants.TOP);
setIcon()	Permet d'assigner une icône
setDisabledIcon()	Permet d'assigner une icône dans un état désactivée

L'alignement vertical par défaut d'un JLabel est centré. L'alignement horizontal par défaut est soit à droite si il ne contient que du texte, soit centré si il contient un image avec ou sans texte. Pour modifier cet alignement, il suffit d'utiliser les méthodes ci dessus en utilisant des constantes en paramètres : SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT, SwingConstants.TOP, SwingConstants.BOTTOM

Par défaut, un JLabel est transparent : son fond n'est pas dessiné. Pour le dessiner, il faut utiliser la méthode setOpaque() :

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.*;

public class TestJLabel2 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");

    f.setSize(100,200);
    JPanel pannel = new JPanel();

    JLabel jLabel1 =new JLabel("Mon texte dans JLabel 1");
    jLabel1.setBackground(Color.red);
    pannel.add(jLabel1);

    JLabel jLabel2 =new JLabel("Mon texte dans JLabel 2");
    jLabel2.setBackground(Color.red);
    jLabel2.setOpaque(true);
    pannel.add(jLabel2);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Dans l'exemple, les 2 JLabel ont le fond rouge demandé par la méthode setBackground(). Seul le deuxième affiche un fond rouge car il est rendu opaque avec la méthode setOpaque().

Il est possible d'associer un raccourci clavier au JLabel qui permet de donner le focus à un autre composant. La méthode setDisplayedMnemonic() permet de définir le raccourci clavier. Celui ci sera activé en utilisant la touche Alt avec le caractère fourni en paramètre. La méthode setLabelFor() permet d'associer le composant fourni en paramètre au raccourci.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.*;

public class TestJLabel3 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    JButton bouton = new JButton("saisir");
    pannel.add(bouton);

    JTextField jEdit = new JTextField("votre nom");

    JLabel jLabel1 =new JLabel("Nom : ");
    jLabel1.setBackground(Color.red);
    jLabel1.setDisplayedMnemonic('n');
    jLabel1.setLabelFor(jEdit);
    pannel.add(jLabel1);
    pannel.add(jEdit);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

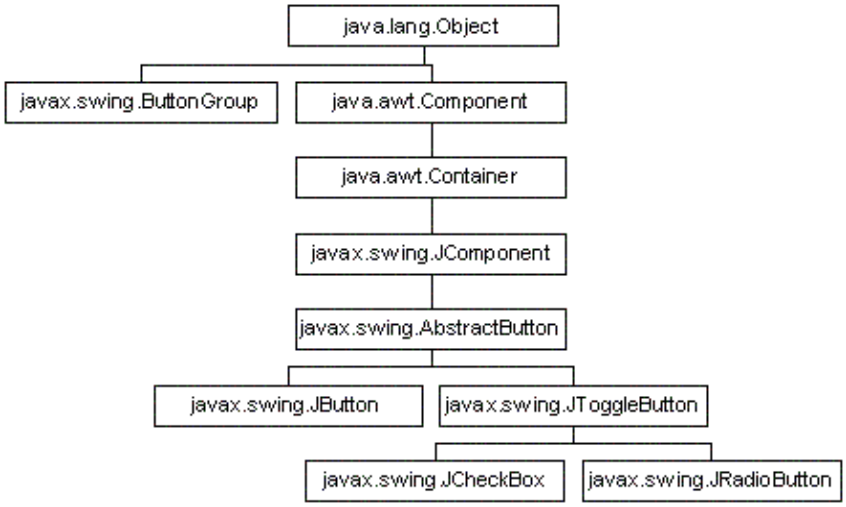
Dans l'exemple, à l'ouverture de la fenêtre, le focus est sur le bouton. Un appui sur Alt+'n' donne le focus au champ de saisie.

14.4.3. Les panneaux : la classe Jpanel

La classe JPanel est un conteneur utilisé pour regrouper et organiser des composants grâce à un gestionnaire de présentation (layout manager). Le gestionnaire par défaut d'un JPanel est un objet de la classe FlowLayout.

14.5. Les boutons

Il existe plusieurs boutons définis par Swing.



14.5.1. La classe AbstractButton

C'est une classe abstraite dont hérite les boutons Swing JButton, JMenuItem et JToggleButton.

Cette classe définit de nombreuses méthodes dont les principales sont :

Méthode	Rôle
AddActionListener	Associer un écouteur sur un évenement de type ActionEvent
AddChangeListener	Associer un écouteur sur un évenement de type ChangeEvent
AddItemListener	Associer un écouteur sur un évenement de type ItemEvent
doClick()	Déclencher un clic par programmation
getText()	Obtenir le texte affiché par le composant
setDisabledIcon()	Associer une icône affichée lorsque le composant à l'état désélectionné
setDisabledSelectedIcon()	Associer une icône affichée lors du passage de la souris sur le composant à l'état désélectionné
setEnabled()	Activer/désactiver le composant
setMnemonic()	Associer un raccourci clavier
setPressedIcon()	Associer une icône affichée lorsque le composant est cliqué
setRolloverIcon()	Associer une icône affichée lors du passage de la souris sur le composant
setRolloverSelectedIcon()	Associer une icône affichée lors du passage de la souris sur le composant à l'état sélectionné
setSelectedIcon()	Associer une icône affichée lorsque le composant à l'état sélectionné

setText()	Mettre à jour le texte du composant
isSelected()	Indiquer si le composant est dans l'état sélectionné
setSelected()	Mettre à jour l'état sélectionné du composant

Tous les boutons peuvent afficher du texte et/ou une image.

Il est possible de préciser une image différente lors du passage de la souris sur le composant et lors de l'enfoncement du bouton : dans ce cas, il faut créer trois images pour chacun des états (normal, enfoncé et survolé). L'image normale est associée au bouton grâce au constructeur, l'image enfoncée grâce à la méthode setPressedIcon() et l'image lors d'un survole grâce à la méthode setRolloverIcon(). Il suffit enfin d'appeler la méthode setRolloverEnable() avec en paramètre la valeur true.

Exemple (code Java 1.1) :

```
import javax.swing.*;
import java.awt.event.*;

public class swing4 extends JFrame {

   public swing4() {
     super("titre de l'application");

     WindowListener l = new WindowAdapter() {
       public void windowClosing(WindowEvent e){
         System.exit(0);
       }
     };
     addWindowListener(l);

     ImageIcon imageNormale = new ImageIcon("arrow.gif");
     ImageIcon imagePassage =  new ImageIcon("arrowr.gif");
     ImageIcon imageEnfoncee = new ImageIcon("arrowy.gif");

     JButton bouton = new JButton("Mon bouton",imageNormale);
     bouton.setPressedIcon(imageEnfoncee);
     bouton.setRolloverIcon(imagePassage);
     bouton.setRolloverEnabled(true);
     getContentPane().add(bouton, "Center");

     JPanel panneau = new JPanel();
     panneau.add(bouton);
     setContentPane(panneau);
     setSize(200,100);
     setVisible(true);
   }

   public static void main(String [] args){
      JFrame frame = new swing4();
   }
}
```

Un bouton peut recevoir des évenements de type ActionEvents (le bouton a été activé), ChangeEvents, et ItemEvents.

Exemple (code Java 1.1) : fermeture de l'application lors de l'activation du bouton

```
import javax.swing.*;
import java.awt.event.*;

public class TestJButton3 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();
```

```
    JButton bouton1 = new JButton("Bouton1");
    bouton1.addActionListener( new ActionListener() {
       public void actionPerformed(ActionEvent e) {
         System.exit(0);
       }
    }
    );

    pannel.add(bouton1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Pour de plus amplements informations sur la gestion des événements, voir le chapitre correspondant.

14.5.2. La classe JButton

JButton est un composant qui représente un bouton : il peut contenir un texte et/ou une icône.

Les constructeurs sont :

Constructeur	Rôle
JButton()	
JButton(String)	préciser le texte du bouton
JButton(Icon)	préciser une icône
JButton(String, Icon)	préciser un texte et une icone

Il ne gère pas d'état. Toutes les indications concernant le contenu du composant JLabel sont valables pour le composant JButton.

Exemple (code Java 1.1) : un bouton avec une image

```
import javax.swing.*;
import java.awt.event.*;

public class swing3 extends JFrame {

   public swing3() {

      super("titre de l'application");

      WindowListener l = new WindowAdapter() {
         public void windowClosing(WindowEvent e){
            System.exit(0);
         }
      };
      addWindowListener(l);

      ImageIcon img = new ImageIcon("tips.gif");
      JButton bouton = new JButton("Mon bouton",img);

      JPanel panneau = new JPanel();
      panneau.add(bouton);
      setContentPane(panneau);
      setSize(200,100);
      setVisible(true);
  }

  public static void main(String [] args){
      JFrame frame = new swing3();
```

```
  }

}
```

L'image gif peut être une animation.

Dans un conteneur de type JRootPane, il est possible de définir un bouton par défaut grace à sa méthode setDefaultButton().

Exemple (code Java 1.1) : définition d'un bouton par défaut dans un JFrame

```
import javax.swing.*;
import java.awt.*;

public class TestJButton2 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();
    JButton bouton1 = new JButton("Bouton 1");
    pannel.add(bouton1);

    JButton bouton2 = new JButton("Bouton 2");
    pannel.add(bouton2);

    JButton bouton3 = new JButton("Bouton 3");
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.getRootPane().setDefaultButton(bouton3);
    f.setVisible(true);
  }
}
```

Le bouton par défaut est activé par un appui sur la touche Entrée alors que le bouton actif est activé par un appui sur la barre d'espace.

La méthode isDefaultButton() de JButton permet de savoir si le composant est le bouton par défaut.

14.5.3. La classe JToggleButton

Cette classe définit un bouton à deux états : c'est la classe mère des composants JCheckBox et JRadioButton.

La méthode setSelected() héritée de AbstractButton permet de mettre à jour l'état du bouton. La méthode isSelected() permet de connaître cet état.

14.5.4. La classe ButtonGroup

La classe ButtonGroup permet de gérer un ensemble de boutons en garantissant qu'un seul bouton du groupe sera sélectionné.

Pour utiliser la classe ButtonGroup, il suffit d'instancier un objet et d'ajouter des boutons (objets héritant de la classe AbstractButton) grâce à la méthode add(). Il est préférable d'utiliser des objets de la classe JToggleButton ou d'une de ces classes filles car elles sont capables de gérer leurs états.

Exemple (code Java 1.1) :

```
import javax.swing.*;

public class TestGroupButton1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    ButtonGroup groupe = new ButtonGroup();
    JRadioButton bouton1 = new JRadioButton("Bouton 1");
    groupe.add(bouton1);
    pannel.add(bouton1);
    JRadioButton bouton2 = new JRadioButton("Bouton 2");
    groupe.add(bouton2);
    pannel.add(bouton2);
    JRadioButton bouton3 = new JRadioButton("Bouton 3");
    groupe.add(bouton3);
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

14.5.5. Les cases à cocher : la classe JCheckBox

Les constructeurs sont les suivants :

Constructeur	Rôle
JCheckBox(String)	précise l'intitulé
JCheckBox(String, boolean)	précise l'intitulé et l'état
JCheckBox(Icon)	précise une icône comme intitulé
JCheckBox(Icon, boolean)	précise une icône comme intitulé et l'état
JCheckBox(String, Icon)	précise un texte et une icône comme intitulé
JCheckBox(String, Icon, boolean)	précise un texte et une icône comme intitulé et l'état

Un groupe de cases à cocher peut être défini avec la classe ButtonGroup. Dans ce cas, un seul composant du groupe peut être sélectionné. Pour l'utiliser, il faut créer un objet de la classe ButtonGroup et utiliser la méthode add() pour ajouter un composant au groupe.

Exemple (code Java 1.1) :

```
import javax.swing.*;

public class TestJCheckBox1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();

    JCheckBox bouton1 = new JCheckBox("Bouton 1");
    pannel.add(bouton1);
    JCheckBox bouton2 = new JCheckBox("Bouton 2");
    pannel.add(bouton2);
    JCheckBox bouton3 = new JCheckBox("Bouton 3");
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
```

```
    f.setVisible(true);
  }
}
```

14.5.6. Les boutons radio : la classe JRadioButton

Les constructeurs sont les mêmes que ceux de la classe JCheckBox.

Exemple (code Java 1.1) :

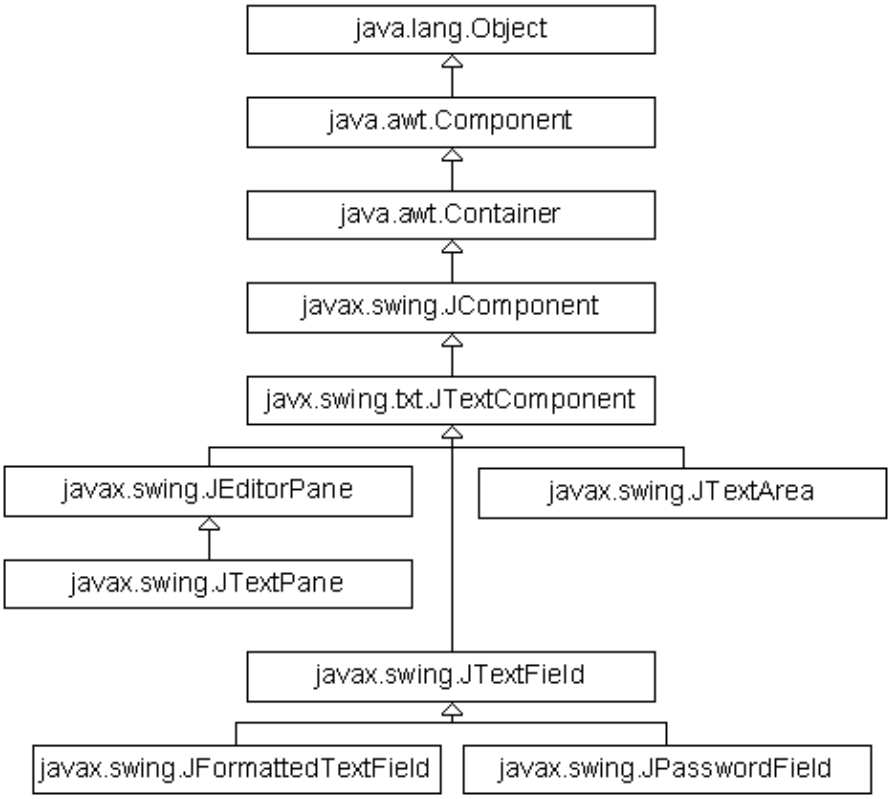
```
import javax.swing.*;

public class TestJRadioButton1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JPanel pannel = new JPanel();
    JRadioButton bouton1 = new JRadioButton("Bouton 1");
    pannel.add(bouton1);
    JRadioButton bouton2 = new JRadioButton("Bouton 2");
    pannel.add(bouton2);
    JRadioButton bouton3 = new JRadioButton("Bouton 3");
    pannel.add(bouton3);

    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Pour regrouper plusieurs boutons radio, il faut utiliser la classe CheckboxGroup

La suite de ce section sera développée dans une version future de ce document

14.6. Les composants de saisie de texte

Swing possède plusieurs composants pour permettre la saisie de texte.



14.6.1. La classe JTextComponent

La classe abstraite JTextComponent est la classe mère de tout les composants permettant la saisie de texte.

Les données du composant (le modèle dans le motif de conception MVC) sont encapsulées dans un objet qui implémente l'interface Document. Deux classes implémentant cette interface sont fournies en standard : PlainDocument pour du texte simple et StyledDocument pour du texte riche pouvant contenir entre autre plusieurs polices de caractères, des couleurs, des images, ...

La classe JTextComponent possède de nombreuses méthodes dont les principales sont :

Méthode	Rôle
void copy()	Copier le contenu du texte et le mettre dans le presse papier système
void cut()	Couper le contenu du texte et le mettre dans le presse papier système
Document getDocument()	Renvoyer l'objet de type Document qui encapsule le texte saisi
String getSelectectedText()	Renvoyer le texte sélectionné dans le composant
int getSelectionEnd()	Renvoyer la position de la fin de la sélection
int getSelectionStart()	Renvoyer la position du début de la sélection
String getText()	Renvoyer le texte saisie
String getText(int, int)	Renvoyer une portion du texte incluse à partir de la position donnée par le premier paramètre et la longueur donnée dans le second paramètre
bool isEditable()	Renvoyer un booléen qui précise si le texte est éditable ou non
void paste()	Coller le contenu du presse papier système dans le composant
void select(int,int)	Sélectioner une portion du texte dont les positions de début et de fin sont

	fournies en paramères
void setCaretPosition(int)	Déplacer la curseur à la position dans le texte précisé en paramètre
void setEditable(boolean)	Permet de préciser si les données du composant sont éditables ou non
void setSelectionEnd(int)	Modifier la position de la fin de la sélection
void setSelectionStart(int)	Modifier la position du début de la sélection
void setText(String)	Modifier le contenu du texte

Toutes ces méthodes sont donc accessibles grâce à l'héritage pour tous les composants de saisie de texte proposés par Swing.

14.6.2. La classe JTextField

La classe javax.Swing.JTextField est un composant qui permet la saisie d'une seule ligne de texte simple. Son modèle utilise un objet de type PlainDocument.

Exemple (code Java 1.1) :

```
import javax.swing.*;

public class JTextField1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JTextField  testField1 = new JTextField ("mon texte");

    pannel.add(testField1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La propriété horizontalAligment permet de préciser l'alignement du texte dans le composant en utilisant les valeurs JTextField.LEFT , JTextField.CENTER ou JTextField.RIGHT.

14.6.3. La classe JPasswordField

La classe JPasswordField permet la saisie d'un texte dont tous les caractères saisis seront affichés sous la forme d'un caractère particulier ('*' par défaut). Cette classe hérite de la classe JTextField.

Exemple (code Java 1.1) :

```
import java.awt.Dimension;

import javax.swing.*;
```

```
public class JPasswordField1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JPasswordField  passwordField1 = new JPasswordField ("");
    passwordField1.setPreferredSize(new Dimension(100,20 ));

    pannel.add(passwordField1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

La méthode setEchoChar(char) permet de préciser le caractère qui sera utilisé pour afficher la saisie d'un caractère.

Il ne faut pas utiliser la méthode getText() qui est déclarée deprecated mais la méthode getPassword() pour obtenir la valeur du texte saisi.

Exemple (code Java 1.1) :

```
import java.awt.Dimension;
import java.awt.event.*;

import javax.swing.*;

public class JPasswordField2 implements ActionListener {

  JPasswordField passwordField1 = null;

  public static void main(String argv[]) {
    JPasswordField2 jpf2 = new JPasswordField2();
    jpf2.init();
  }

  public void init() {
    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    passwordField1 = new JPasswordField("");
    passwordField1.setPreferredSize(new Dimension(100, 20));
    pannel.add(passwordField1);

    JButton bouton1 = new JButton("Afficher");
    bouton1.addActionListener(this);

    pannel.add(bouton1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {

    System.out.println("texte saisie = " + String.copyValueOf(passwordField1.getPassword()));
  }

}
```

Les méthodes copy() et cut() sont redéfinies pour empêcher l'envoi du contenu dans le composant et émettre simplement un beep.

14.6.4. La classe JFormattedTextField

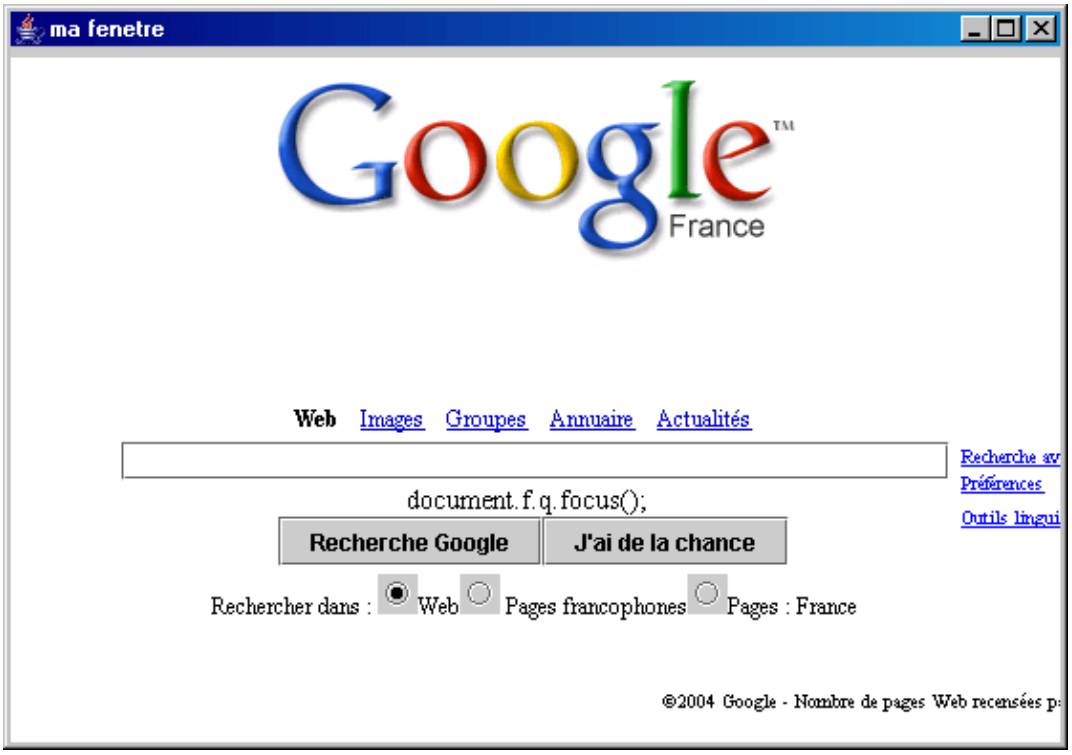
Le JDK 1.4 propose la classe JFormattedTextField pour faciliter la création d'un composant de saisie personnalisé. Cette classe hérite de la classe JTextField.

14.6.5. La classe JEditorPane

Ce composant permet de saise de texte riche multi−lignes. Ce type de texte peut contenir des informations de mise en pages et de formatage. En standard, Swing propose le support des formats RTF et HTML.

Exemple (code Java 1.1) : affichage de la page de Google avec gestion des hyperliens

```
import java.net.URL;
import javax.swing.*;
import javax.swing.event.*;

public class JEditorPane1 {

  public static void main(String[] args) {
    final JEditorPane editeur;
    JPanel pannel = new JPanel();

    try {
      editeur = new JEditorPane(new URL("http://google.fr"));
      editeur.setEditable(false);
      editeur.addHyperlinkListener(new HyperlinkListener() {
        public void hyperlinkUpdate(HyperlinkEvent e) {
          if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
            URL url = e.getURL();
            if (url == null)
              return;
            try {
              editeur.setPage(e.getURL());
            } catch (Exception ex) {
              ex.printStackTrace();
            }
          }
        }
      });

      pannel.add(editeur);
    } catch (Exception e1) {
      e1.printStackTrace();
    }
    JFrame f = new JFrame("ma fenetre");
    f.setSize(500, 300);

    f.getContentPane().add(pannel);
    f.setVisible(true);

  }
}
```



14.6.6. La classe JTextPane

La suite de cette section sera développée dans une version future de ce document

14.6.7. La classe JTextArea

La classe JTextArea est un composant qui permet la saisie de texte simple en mode multi−lignes. Le modèle utilisé par ce composant est le PlainDocument : il ne peut donc contenir que du texte brut sans éléments multiples de formatage.

JTexteArea propose plusieurs méthodes pour ajouter du texte dans son modèle :

- soit fournir le texte en pramètre du constructeur utilisé
- soit utiliser la méthode setText() qui permet d'initialiser le texte du composant
- soit utiliser la méthode append() qui permet d'ajouter du texte à la fin de celui contenu dans le texte du composant
- soit utiliser la méthode insert() permet d'insérer un texte à une position données en caractères dans dans le texte du composant

La méthode replaceRange() permet de remplacer une partie du texte désignée par la position du caractère de début et la position de son caractère de fin par le texte fourni en paramètre.

La propriété rows permet de définir le nombre de ligne affichée par le composant : cette propriété peut donc être modifié lors d'un redimensionnement du composant. La propriété lineCount en lecture seule permet de savoir le nombre de lignes dont le texte est composé. Il ne faut pas confondre ces deux propriétés.

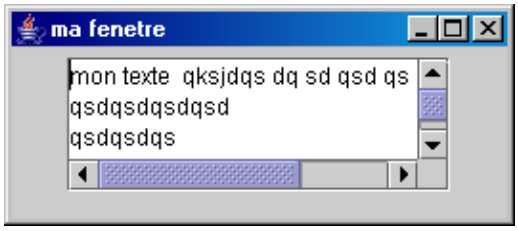
Exemple (code Java 1.1) :

```
import javax.swing.*;

public class JTextArea1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JTextArea  textArea1 = new JTextArea ("mon texte");

    pannel.add(textArea1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

Par défaut, la taille du composant augmente au fur et à mesure de l'augmentation de la taille du texte qu'il contient. Pour éviter cet effet, il faut encapsuler le JTextArea dans un JScrollPane.

Exemple (code Java 1.1) :

```
import java.awt.Dimension;
import javax.swing.*;

public class JTextArea1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();
    JTextArea  textArea1 = new JTextArea ("mon texte");
    JScrollPane scrollPane = new JScrollPane(textArea1);
    scrollPane.setPreferredSize(new Dimension(200,70));
    pannel.add(scrollPane);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```



14.7. Les onglets

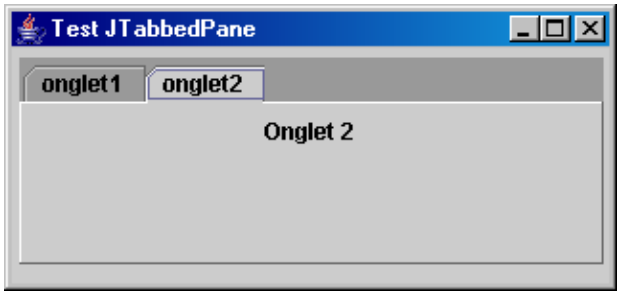
La classe javax.swing.JTabbedPane encapsule un ensemble d'onglets. Chaque onglet est constitué d'un titre, d'un composant et éventuellement d'une image.

Pour utiliser ce composant, il faut :

- instancier un objet de type JTabbedPane
- créer le composant de chaque onglet
- ajouter chaque onglet à l'objet JTabbedPane en utilisant la méthode addTab()

Exemple (code Java 1.1) :

```
import java.awt.Dimension;
import java.awt.event.KeyEvent;

import javax.swing.*;

public class TestJTabbedPane1 {

  public static void main(String[] args) {
    JFrame f = new JFrame("Test JTabbedPane");
    f.setSize(320, 150);
    JPanel pannel = new JPanel();

    JTabbedPane onglets = new JTabbedPane(SwingConstants.TOP);

    JPanel onglet1 = new JPanel();
    JLabel titreOnglet1 = new JLabel("Onglet 1");
    onglet1.add(titreOnglet1);
    onglet1.setPreferredSize(new Dimension(300, 80));
    onglets.addTab("onglet1", onglet1);

    JPanel onglet2 = new JPanel();
    JLabel titreOnglet2 = new JLabel("Onglet 2");
    onglet2.add(titreOnglet2);
    onglets.addTab("onglet2", onglet2);

    onglets.setOpaque(true);
    pannel.add(onglets);
    f.getContentPane().add(pannel);
    f.setVisible(true);

  }
}
```



A partir du JDK 1.4, il est possible d'ajouter un raccourci clavier sur chacun des onglets en utilisant la méthode setMnemonicAt(). Cette méthode attend deux paramètres : l'index de l'onglet concerné (le premier commence à 0) et la touche du clavier associée sous la forme d'une constance KeyEvent.VK_xxx. Pour utiliser ce raccourci, il suffit d'utiliser la touche désignée en paramètre de la méthode avec la touche Alt.

La classe JTabbedPane possède plusieurs méthodes qui permettent de définir le contenu de l'onglets :

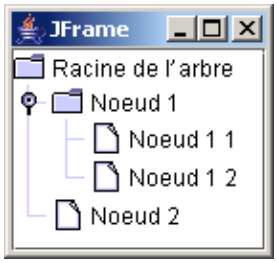
Méthodes	Rôles

addTab(String, Component)	Permet d'ajouter un nouvel onglet dont le titre et le composant sont fournis en paramètres. Cette méthode possède plusieurs surcharges qui permettent de préciser une icône et une bulle d'aide
insertTab(String, Icon, Component, String, index)	Permet d'insérer un onglet dont la position est précisée dans le dernier paramètre
remove(int)	Permet de supprimer l'onglet dont l'index est fourni en paramètre
setTabPlacement	Permet de préciser le positionnement des onglets dans le composant JTabbedPane. Les valeurs possibles sont les constantes TOP, BOTTOM, LEFT et RIGHT définies dans la classe JTabbedPane.

La méthode getSelectedIndex() permet d'obtenir l'index de l'onglet courant. La méthode setSelectedIndex() permet de définir l'onglet courant.

14.8. Le composant JTree

Le composant JTree permet de présenter des données sous une forme hiérarchique arborescente.



Aux premiers abords, le composant JTree peut sembler compliqué à mettre en oeuvre mais la compréhension de son mode de fonctionnement peut grandement failiter son utilisation.

Il utilise le modèle MVC en proposant une séparation des données (data models) et du rendu de ces données (cell renderers).

Dans l'arbre, les éléments qui ne possèdent pas d'élément fils sont des feuilles (leaf). Chaque élément est associé à un objet (user object) qui va permettre de déterminé le libellé affiché dans l'arbre en utilisant la méthode toString().

14.8.1. La création d'une instance de la classe JTree

La classe JTree possède 7 constructeurs dont tous ceux qui attendent au moins un paramètre acceptent une collection pour initialiser tout ou partie du modèle de données de l'arbre :

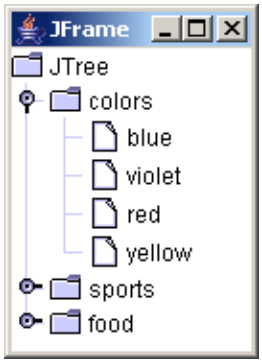
public JTree();
public JTree(Hashtable value);
public JTree(Vector value);
public JTree(Object[] value);
public JTree(TreeModel model);
public JTree(TreeNode rootNode);
public JTree(TreeNode rootNode, boolean askAllowsChildren);

Lorsqu'une instance de JTree est créée avec le constructeur par défaut, l'arbre obtenu contient des données par défaut.

Exemple (code Java 1.1) :

```
import javax.swing.JFrame;
import javax.swing.JTree;
```

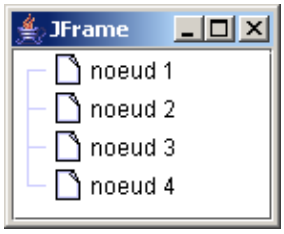
```
public class TestJtree extends JFrame {

  private javax.swing.JPanel jContentPane = null;
  private JTree              jTree        = null;

  private JTree getJTree() {
    if (jTree == null) {
      jTree = new JTree();
    }
    return jTree;
  }

  public static void main(String[] args) {
    TestJtree testJtree = new TestJtree();
    testJtree.setVisible(true);
  }

  public TestJtree() {
    super();
    initialize();
  }

  private void initialize() {
    this.setSize(300, 200);
    this.setContentPane(getJContentPane());
    this.setTitle("JFrame");
  }

  private javax.swing.JPanel getJContentPane() {
    if (jContentPane == null) {
      jContentPane = new javax.swing.JPanel();
      jContentPane.setLayout(new java.awt.BorderLayout());
      jContentPane.add(getJTree(), java.awt.BorderLayout.CENTER);
    }
    return jContentPane;
  }
}
```



Les trois constructeurs qui attendent en paramètre une collection permettent de créer un arbre avec une racine non affichée qui va contenir comme noeuds fils directs tous les éléments contenus dans la collection.

Exemple (code Java 1.1) :

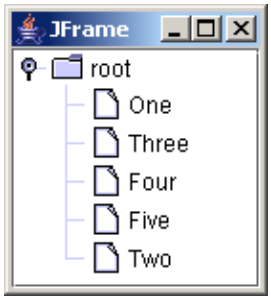
```
      String[] = {"noeud 1","noeud 2","noeud3","noeud 4"};
      jTree = new JTree(racine);
```



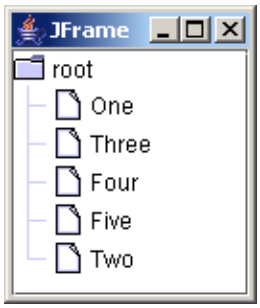
Dans ce cas, la racine n'est pas affichée. Pour l'afficher, il faut utiliser la méthode setRootVisible()

Exemple (code Java 1.1) :

```
jTree.setRootVisible(true);
```



Dans ce cas elle se nomme root et possède un commutateur qui permet de refermer ou d'étendre la racine. Pour supprimer ce commutateur, il faut utiliser la méthode jTree.setShowsRootHandles(false)



L'utilisation de l'une ou l'autre des collections n'est pas équivalente. Par exemple, l'utilisation d'une hashtable ne garantie pas l'ordre des noeuds puisque cette collection ne gère pas par définition un ordre précis.

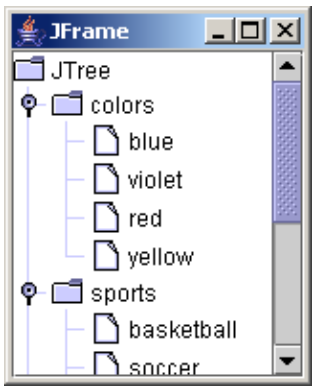
Généralement, la construction d'un arbre utilise un des constructeurs qui attend en paramètre un objet de type TreeModel ou TreeNode car ces deux objets permettent d'avoir un contrôle sur l'ensemble des données de l'arbre. Leur utilisation sera détaillée dans la section consacrée à la gestion des données de l'arbre

En fonction du nombre d'éléments et de l'état étendue ou non d'un ou plusieurs éléments, la taille de l'arbre peut varier : il est donc nécesssaire d'inclure le composant JTree dans un composant JScrollPane

Exemple (code Java 1.1) :

```
...

  private JScrollPane        jScrollPane  = null;

...

  private JScrollPane getJScrollPane() {
    if (jScrollPane == null) {
      jScrollPane = new JScrollPane();
      jScrollPane.setViewportView(getJTree());
    }
    return jScrollPane;
  }

...

  private javax.swing.JPanel getJContentPane() {
    if (jContentPane == null) {
      jContentPane = new javax.swing.JPanel();
      jContentPane.setLayout(new java.awt.BorderLayout());
      jContentPane.add(getJScrollPane(), java.awt.BorderLayout.CENTER);
    }
    return jContentPane;
```

```
  }
```



L'utilisateur peut sélectionner un noeud en cliquant sur son texte ou son icône. Un double clic sur le texte ou l'icône d'un noeud permet de l'étendre ou le refermer selon son état.

14.8.2. La gestion des données de l'arbre

Chaque arbre commence par un noeud racine. Par défaut, la racine et ces noeuds fils directs sont visibles. Chaque noeud de l'arbre peut avoir zéro ou plusieurs noeuds fils. Un noeud sans noeud fils est appelé un feuille de l'arbre (leaf)

En application du modèle MVC, le composant JTree ne gère pas directement chaque noeud et la façon dont ceux−ci sont organiser et stocker mais il utilise un objet dédié de type TreeModel.

Ainsi, comme dans d'autre composant Swing, le composant JTree manipule des objets implémentant des interfaces. Une classe qui encapsule les données de l'arbre doit implémenter l'interface TreeModel. Chaque noeud de l'arbre doit implémenter l'interface TreeNode.

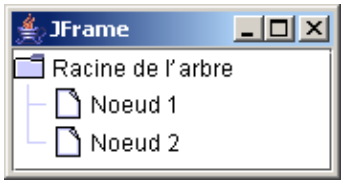
Pour préciser les données contenues dans l'arbre, il faut créer un objet qui va encapsuler ces données et les passer au constructeur de la classe Jtree. Cet objet peut être de type TreeNode ou TreeModel. Cet objet stocke les données de chaque noeud dans un objet de type TreeNode.

Généralement, le plus simple est de définir un type TreeNode personnalisé. Swing propose pour cela l'objet DefaultMutableTreeNode. Donc le plus simple est de créer une instance de type DefaultMutableTreeNode pour stocker les données et l'utiliser lors de l'appel du constructeur de la classe JTree.

La classe DefaultMutableTreeNode implémente l'interface MutableTreeNode qui elle−même hérite de l'interface TreeNode

Exemple (code Java 1.1) :

```
import javax.swing.tree.DefaultMutableTreeNode;

...

  private JTree getJTree() {

    if (jTree == null) {
      DefaultMutableTreeNode racine = new DefaultMutableTreeNode("Racine de l'arbre");
      DefaultMutableTreeNode noeud1 = new DefaultMutableTreeNode("Noeud 1");
      racine.add(noeud1);
      DefaultMutableTreeNode noeud2 = new DefaultMutableTreeNode("Noeud 2");
      racine.add(noeud2);
      jTree = new JTree(racine);
    }
    return jTree;
  }
```



Dans ce cas, une instance de la classe DefaultTreeModel est créée avec la racine fournie en paramètre du constructeur de la classe JTree.

Une autre solution permet de créer une instance de la classe DefaultTreeModel et de la passer en paramètre du constructeur de la classe JTree.

La méthode setModel() de la classe JTree permet d'associer un modèle de données à l'arbre.

14.8.2.1. L'interface TreeNode

Chaque noeud de l'arbre stocké dans le modèle de données implémente l'interface TreeNode.

Cette interface définit 7 méthodes dont la plupart concernent les relations entre les noeuds :

Méthode	Rôle
Enumeration children()	renvoie une collection des noeuds fils
boolean getAllowsChildren()	Renvoie un booléen qui precise si le noeud peut avoir des noeuds fils
TreeNode getChildAt(int index)	Renvoie le noeud fils correspondant à l'index fourni en paramètre
int getChildCount()	renvoie le nombre de noeuds fils direct du noeud
int getIndex(TreeNode child)	renvoie l'index du noeud passé en paramètre
TreeNode getParent()	renvoie le noeud père
boolean isLeaf()	renvoie un booléen qui précise si le noeud est une feuille

Chaque noeud ne peut avoir qu'un seul père (hormis le noeud racine qui ne possède pas de père) et autant de noeuds fils que souhaité. La méthode getParent() permet de renvoyer le noeud père. Elle renvoie null lorsque cette méthode est appelée sur le noeud racine.

La méthode getChildCount() renvoie le nombre de noeuds fils direct du noeud.

Si la méthode getAllowsChildren() permet de préciser si le noeud peut avoir des noeuds enfants : elle renvoie false alors le noeud sera toujours une feuille et ne pourra donc jamais avoir de noeuds fils.

La méthode isLeaf() renvoie un booléen précisant si le noeud est une feuille ou non. Une feuille est un noeud qui ne possède pas de noeud fils.

Les noeuds fils sont ordonnés car l'ordre de représentation des données peut être important dans la représentation de données hiérarchiques. La méthode getChildAt() renvoie le noeud fils dont l'index est fourni en paramètre de la méthode. La méthode getIndex() renvoie l'index du noeud fils passé en paramètre .

14.8.2.2. L'interface MutableTreeNode

Les 7 méthodes définies par l'interface TreeNode ne permettent que de lire des valeurs. Pour mettre à jour un noeud, il est nécessaire d'utiliser l'interface MutableTreeNode qui hérite de la méthode TreeNode. Elle définit en plus plusieurs méthodes permettant de mettre à jour le noeud.

void insert(MutableTreeNode child, int index);
void remove(int index);
void remove(MutableTreeNode node);
void removeFromParent();
void setParent(MutableTreeNode parent);
void setUserObject(Object userObject);

La méthode insert() permet d'ajouter le noeud fourni en paramètre comme noeud fils à la position précisée par le second paramètre.

Il existe deux surcharges de la méthode remove() qui permet de déconnecter un noeud fils de son père. La première surcharge attend en paramètre l'index du noeud fils. La seconde surcharge attend en paramètre le noeud à déconnecter. Dans tout les cas, il est nécessaire d'utiliser cette méthode sur le noeud père.

La méthode removeFromParent() appelée à partir d'un noeud permet de supprimer le lien entre le noeud et son père.

La méthode setParent() permet de préciser le père du noeud.

La méthode setUserObject() permet d'associer un objet au noeud. L'appel à la méthode toString() de cet objet permettra de déterminer la libellé du noeud qui sera affiché.

14.8.2.3. La classe DefaultMutableTreeNode

Généralement, les noeuds créés dans le modèle sont des instances de la classe DefaultMutableTreeNode. Cette classe implémente l'interface MutableTreeNode ce qui permet d'obtenir une instance d'un noeud modifiable.

Généralement, les noeuds fournis en paramètres des méthodes proposées par Swing sont de type TreeNode. Si l'instance du noeud est de type DefaultTreeNode, il est possible de faire un cast pour accéder à toutes ces méthodes.

La classe propose trois constructeurs dont deux attendent en paramètre l'objet qui sera associé au noeud. L'un des deux attend en plus un booléen qui permet de préciser si le noeud peut avoir des noeuds fils.

Constructeur	Rôle
public DefaultMutableTreeNode()	créé un noeud sans objet asssocié. Cette association pourra être faite avec la méthode setObject()
public DefaultMutableTreeNode(Object userObject)	créé un noeud en précisant l'objet qui lui sera associé et qui pourra avoir des noeuds fils
public DefaultMutableTreeNode(Object userObject, boolean allowsChildren)	créé un noeud dont le booléen précise si il pourra avoir des fils

Pour ajouter une instance de la classe DefaultMutableTreeNode dans le modèle de l'arbre, il est possible d'utiliser la méthode insert() de l'interface MutuableTreeNode ou utiliser la méthode add() de la classe DefaultMutableTreeNode. Celle−ci attend en paramètre une instance du noeud fils à ajouter. Elle ajoute le noeud après le dernier noeud fils, ce qui évite d'avoir à garder une référence sur la position où insérer le noeud.

Exemple (code Java 1.1) :

```
DefaultMutableTreeNode racineNode                = new DefaultMutableTreeNode();
DefaultMutableTreeNode division1 =                 new DefaultMutableTreeNode("Division 1");
DefaultMutableTreeNode division2 =                 new DefaultMutableTreeNode("Division 2");
racineNode.add(division1);
racineNode.add(division2);
jTree.setModel(new DefaultTreeModel(racineNode));
```

Il est aussi possible de définir sa propre classe qui implémente l'interface MutableTreeNode : une possiblité est de définir une classe fille de la classe DefaultMutableTreeNode.

14.8.3. La modification du contenu de l'arbre

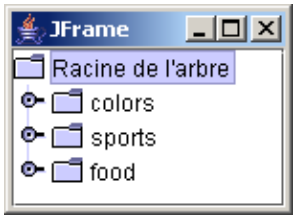
Les modifications du contenu de l'arbre peuvent se faire au niveau du modèle (DefaultTreeModel) ou au niveau du noeud.

La méthode getModel() de la classe JTree permet d'obtenir une référence sur l'instance de la classe TreeModel qui encapsule le modèle de données.

Il est ainsi possible d'accéder à tous les noeuds du modèle pour les modifier.

Exemple (code Java 1.1) : modifier le contenu de la racine de l'arbre

```
jTree = new JTree();
Object noeudRacine = jTree.getModel().getRoot();
((DefaultMutableTreeNode)noeudRacine).setUserObject("Racine de l'arbre");
```



L'interface TreeModel ne propose rien pour permettre la mise à jour du modèle. Pour mettre à jour le modèle, il faut utiliser une instance de la classe DefaultTreeModel.

Elle propose plusieurs méthodes pour ajouter ou supprimer un noeud :

void insertNodeInto(MutableTreeNode child, MutableTreeNode parent, int index)
void removeNodeFromParent(MutableTreeNode parent)

L'avantage de ces deux methodes est qu'elles mettent à jour le modèle mais aussi elles mettent à jour la vue en appelant respectivement les méthodes nodesWereInserted() and nodesWereRemoved() de la classe DefaultTreeModel.

Ces deux méthodes sont donc pratiques pour faire des mises à jour mineures mais elles sont peut adaptées pour de nombreuses mises à jour puisque qu'elles lèvent un événement à chacune de leur utilisation.

14.8.3.1. Les modifications des noeuds fils

La classe DefaultMutableTreeNode propose plusieurs methodes pour metre à jour le modèle à partir du noeud qu'elle encapsule.

void add(MutableTreeNode child)
void insert(MutableTreeNode child, int index)
void remove(int index)
void remove(MutableTreeNode child)
void removeAllChildren()
void removeFromParent()

Toutes ces methodes sauf la dernière agissent sur un ou plusieurs noeuds fils. Ces méthodes agissent simplement sur la structure du modèle. Elles ne provoquent pas un affichage par la partie vue de ces changements dans le modèle. Il est nécessaire d'utiliser une des méthodes suivantes proposées par la classe DefaultTreeModel :

Méthode	Rôle
void reload()	rafraichir toute l'arborscence à partir du modèle

void reload(TreeNode node)	rafraichir toute l'arborescence à partir du noeud précisé en paramètre
void nodesWereInserted(TreeNode node, int[] childIndices)	cette méthode rafraichit pour le noeud précisé les noeuds fils ajoutés dont les index sont fournis en paramètre
void nodesWereRemoved(TreeNode node,int[] childIndices, Object[] removedChildren)	cette méthode rafraichit pour le noeud précisé les noeuds fils supprimés dont les index sont fournis en parmaètre
void nodeStructureChanged(TreeNode node)	cette méthode est identique à la méthode reload()

14.8.3.2. Les événements émis par le modèle

Il est possible d'enregistrer un listener de type TreeModelListener sur un objet de type DefaultTreeModel.

L'interface TreeModelListener définit quatre méthodes pour répondre à des événements particuliers :

Méthode	Rôle
void treeNodesChanged(TreeModelEvent)	la méthode nodeChanged() ou nodesChanged() est utilisée
void treeStructureChanged(TreeModelEvent)	la méthode reload() ou nodeStructureChanged() est utilisée
void treeNodesInserted(TreeModelEvent)	la méthode nodeWhereInserted() est utilisée
void treeNodesRemoved(TreeModelEvent)	la méthode nodeWhereRemoved() est utilisée

Toutes ces méthodes ont un objet de type TreeModelEvent qui encapsule l'événement.

La classe TreeModelEvent encapsule l'événement et propose cinq methodes pour obtenir des inforamtions sur les noeuds impactés par l'événement.

Méthode	Rôle
Object getSource()	renvoie une instance sur le modèle de l'arbre (généralement un objet de type DefaultTreeModel)
TreePath getTreePath()	renvoie le chemin du noeud affecté par l'événement
Object[] getPath()	renvoie le chemin du noeud affecté par l'événement
Object[] getChildren()	
int[] getChildIndices()	

Dans la méthode treeStructureChanged(), seules les methodes getPath() et getTreePath() fournissent des informations utiles en retournant le noeud qui a été modifié.

Dans la méthode treeNodesChanged(), treeNodesRemoved() et treeNodesInserted() les methodes getPath() et getTreePath() renvoient le noeud père des noeuds affectés. Les méthodes getChildIndices() et getChidren() renvoie respectivement un tableau des index des noeuds fils modifiés et un tableau de ces noeuds fils.

Dans les methodes, les methodes getPath() et getTreePath() renvoient le noeud père des noeuds affectés.

Comme l'objet JTree enregistre ces propres listener, il n'est pas nécessaire la plupart du temps, d'enregistrer ces listeners hormis pour des besoins spécifiques.

14.8.3.3. L'édition d'un noeud

Par défaut, le composant JTree est readonly. Il est possible d'autoriser l'utilisateur à modifier le libellé des noeuds en utilisant la méthode setEditable() avec le paramètre true. jTtree.setEditable(true);

Pour éditer un noeud, il faut

- sur un noeud non sélectioné : cliquer rapidement trois fois sur le noeud à modifier
- sur un noeud déjà sélectionné : cliquer une fois sur le noeud ou appuyer sur la touche F2

Pour valider les modifications, il suffit d'appuyer sur la touche « Entree ».

Pour annuler les modifications, il suffit d'appuyer sur la touche « Esc »

Il est possible d'enregistrer un listener de type TreeModelListerner pour assurer des traitements lors d'événements liés à l'édition d'un noeud.

L'interface TreeModelListener définit la méthode treeNodesChanged() qui permet de traiter les événements de type TreeModelEvent liés à la modification d'un noeud.

Exemple (code Java 1.1) :

```
jTree.setEditable(true);
jTree.getModel().addTreeModelListener(new TreeModelListener() {

  public void treeNodesChanged(TreeModelEvent evt) {
    System.out.println("TreeNodesChanged");
    Object[] noeuds = evt.getChildren();
    int[] indices = evt.getChildIndices();
    for (int i = 0; i < noeuds.length; i++) {
      System.out.println("Index " + indices[i] + ", nouvelle valeur : "
          + noeuds[i]);
    }
  }

  public void treeStructureChanged(TreeModelEvent evt) {
    System.out.println("TreeStructureChanged");
  }

  public void treeNodesInserted(TreeModelEvent evt) {
    System.out.println("TreeNodesInserted");
  }

  public void treeNodesRemoved(TreeModelEvent evt) {
    System.out.println("TreeNodesRemoved");
  }

});
```

14.8.3.4. Les éditeurs personnalisés

Il est possible de définir un éditeur particulier pour éditer la valeur d'un noeud. Un éditeur particulier doit implémenter l'interface TreeCellEditor.

Cette interface hérite de l'interface CellEditor qui définie plusieurs méthodes utiles pour la définition d'un éditeur dédié :

Object getCellEditorValue();
boolean isCellEditable(EventObject);
boolean shouldSelectCell(EventObject);
boolean stopCellEditing();
void cancelCellEditing();
void addCellEditorListener(CellEditorListener);
void removeCellEditorListener(CellEditorListener);

L'interface TreeCellEditor ne définie qu'une seule méthode :

Component getTreeCellEditorComponent(JTree tree, Object value, boolean isSelected, boolean expanded, boolean leaf, int row);

Cette méthode renvoie un composant qui va permettre l'éditeur de la valeur du noeud.

La valeur initiale est fournie dans le second paramètre de type Object. Les trois arguments de type booléen suivants permettent respectivement de savoir si le noeud est sélectionné, est étendu et est une feuille.

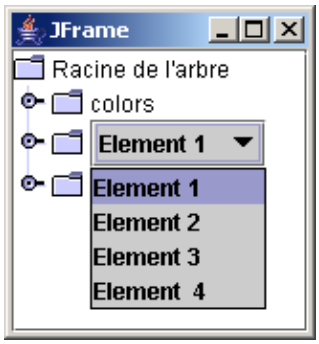
Swing propose une implémentation de cette interface dans la classe DefaultCellEditor qui permet de modifier la valeur du noeud sous la forme d'une zone de texte, d'une case à cocher ou d'une liste déroulante grâce à trois constructeurs :

public DefaultCellEditor(JTextField text); public DefaultCellEditor(JCheckBox box); public DefaultCellEditor(JComboBox combo);

La méthode setCellEditor() de la classe JTree permet d'associer le nouvel éditeur à l'arbre.

Exemple (code Java 1.1) :

```
jTree.setEditable(true);
String[] elements = { "Element 1", "Element 2", "Element 3", "Element  4"};
JComboBox jCombo = new JComboBox(elements);
DefaultTreeCellEditor editor = new DefaultTreeCellEditor(jTree,
        new DefaultTreeCellRenderer(), new DefaultCellEditor(jCombo));
jTree.setCellEditor(editor);
```



14.8.3.5. 3.5 Définir les noeuds éditables

Par défaut, si la méthode setEditable(true) est utilisée alors tous les noeuds sont modifiables.

Il est possible de définir quels sont les noeuds de l'arbre qui sont éditables en créant une classe fille de la classe JTree et en redéfinissant la méthode isPathEditable().

Cette méthode est appelée avant chaque édition d'un noeud. Elle attend en paramètre un objet de type TreePath qui encapsule le chemin du noeud à éditer.

Par défaut, elle renvoie le résutlat de l'appel à la méthode isEditable(). Il est d'ailleurs important, lors de la rédéfinition de méthode isPathEditable(), de tenir compte du résultat de la méthode isEditable() pour s'assurer que l'arbre est modifiable avant vérifier si le noeud peut être modifié.

14.8.4. La mise en oeuvre d'actions sur l'arbre

14.8.4.1. Etendre ou refermer un noeud

Pour étendre un noeud et ainsi voir ces fils, l'utilisateur peut double cliquer sur l'icône ou sur le libellé du noeud. Il peut aussi cliquer sur le petit commutateur à gauche de l'icône.

Enfin, il est possible d'utiliser le clavier pour naviguer dans l'arbre à l'aide des touches flèches haut et bas et des touches flèches droite et gauche pour respectivement éttendre ou refermer un noeud. Lors d'un appui sur la flèche gauche et que le noeud est déjà fermé alors c'est le noeud père qui est sélectionné. De la même façon, lors d'un appui sur la flèche droite et que le noeud est étendu alors le premier noeud fils est sélectionné.

La touche HOME permet de sélectionner le noeud racine. La touche END permet de sélectionner le noeud qui est la dernière feuille du dernier noeud. Les touches PAGEUP et PAGEDOWN permettent de paginer dans les noeuds de l'arbre.

Depuis Java 2 version 1.3, la méthode setToggleClickCount() permet de préciser le nombre de clic nécessaire pour réaliser l'opération pour étendre ou refermer un noeud.

La classe JTree propose plusieurs méthodes liées aux actions permettant d'etendre ou de refermer un noeud

Méthode	**Rôle**
public void expandRow (int row)	Etendre le noeud dont l'index est fourni en paramètre
public void collapseRow(int row)	Refermer le noeud dont l'index est fourni en paramètre
public void expandPath(TreePath path)	Etendre le noeud encapsulé dans la classe TreePath fourni en paramètre
public void collapsePath(TreePath path)	Refermer le noeud encapsulé dans la classe TreePath fourni en paramètre
public boolean isExpanded(int row)	Renvoie un booléen qui précise si le noeud dont l'index est fourni en paramètre est éttendu
public boolean isCollapsed (int row)	Renvoie un booléen qui precise si le noeud dont l'index est fourni en paramètre est refermé
public boolean isExpanded(TreePath path)	Renvoie un booléen qui précise si le noeud encapsulé dans la classe TreePath fourni en paramètre est éttendu
public boolean isCollapsed (TreePath path)	Renvoie un booléen qui précise si le noeud encapsulé dans la classe TreePath fourni en paramètre est refermé

Par défaut, le noeud racine est étendu.

Les méthodes expandRow() et expandPath() ne permettent que d'étendre les noeuds fils direct du noeud sur lequelle elles sont appliquées. Pour étendre les noeuds sous jacent il est nécessaire d'écrire du code pour réaliser l'opération sur chaque noeud concerné de façon récursive.

Pour refermer tous les noeux et ne laisser que le noeud racine, il faut utiliser la méthode collapseRow() et lui passant 0 comme paramètre puisque le noeud racine est toujours le premier noeud.

Exemple (code Java 1.1) :

```
jTree.collapseRow(0);
```

La classe JTree propose deux méthodes pour forcer un noeud à être visible : scrollPathToVisible() et scrollRowToVisible(). Celles−ci ne peuvent fonctionner que si le composant JTree est inclus dans un conteneur JScrollPane pour permettre au composant de scroller.

Exemple (code Java 1.1) :

```
     jTree.addTreeExpansionListener(new TreeExpansionListener() {
       public void treeExpanded(TreeExpansionEvent evt) {
         System.out.println("treeExpanded : path=" + evt.getPath());
         jTree.scrollPathToVisible(evt.getPath());
       }
```

14.8.4.2. Déterminer le noeud sélectionné

Pour déterminer le noeud sélectionné, il suffit d'utiliser la méthode getLastSelectedPathComponent() de la classe JTree et de caster la valeur retournée dans le type du noeud, généralement de type DefaultMutableTreeNode. La méthode getObject() du noeud permet d'obtenir l'objet associé au noeud. Si l'objet associé est simplement une chaîne de caractères ou si la valeur nécessaire est simplement le libellé du noeud, il suffit d'utiliser la méthode toString().

Exemple (code Java 1.1) : un bouton qui précise lors d'un clic le noeud sélectionné

```
...
 private JButton getJButton() {
   if (jButton == null) {
     jButton = new JButton();
     jButton.addActionListener(new java.awt.event.ActionListener() {
       public void actionPerformed(java.awt.event.ActionEvent e) {
         System.out.println("actionPerformed()");
         System.out.println("Noeud sélectionné : "
             + jTree.getLastSelectedPathComponent().toString());
       }
     });
   }
```

14.8.4.3. Parcourir les noeuds de l'arbre

Il peut être nécessaire de parcourir tout ou partie des noeuds de l'arbre pour par exemple faire une recherche dans l'arborescence.

Si l'arbre est composé de noeuds de type DefaultMutableTreenode alors l'interface TreeNode propose plusieurs méthodes pour obtenir une énumération des noeuds pour parcourir tout ou partie de l'arboresence dans les deux sens et deux ordres :

Enumeration preorderEnumeration();
Enumeration postorderEnumeration();
Enumeration breadthFirstEnumeration();
Enumeration depthFirstEnumeration();

Dans l'exemple ci−dessous, l'arborescence suivante est utilisée :

Exemple (code Java 1.1) : un bouton qui précise lors d'un clic le noeud sélectionné

```
Enumeration e = ((DefaultMutableTreeNode)jTree.getModel().getRoot()).preorderEnumeration();
 while (e.hasMoreElements()) {
    System.out.println(e.nextElement() + " ");
 }
```

Résultat :

preorder	postorder	breadthFirst	depthFirst
Racine de l'arbre	blue	Racine de l'arbre	blue
colors	violet	colors	violet
blue	red	sports	red
violet	yellow	food	yellow
red	colors	blue	colors
yellow	basketball	violet	basketball
sports	soccer	red	soccer
basketball	football	yellow	football
soccer	hockey	basketball	hockey
football	sports	soccer	sports
hockey	hot dogs	football	hot dogs
food	pizza	hockey	pizza
hot dogs	ravioli	hot dogs	ravioli
pizza	bananas	pizza	bananas
ravioli	food	ravioli	food
bananas	Racine de l'arbre	bananas	Racine de l'arbre

La méthode pathFromAncestorEnumeration(TreeNode ancestor)renvoie une énumération des noeuds entre le noeud sur lequel la méthode est appelée et le noeud fourni en paramètre. Ainsi le noeud fourni en paramètre doit obligatoirement être un noeud fils direct ou indirect du noeud sur lequel la méthode est appelée. Dans le cas contraire, une exception de type IllegalArgumentException est levée.

14.8.5. La gestion des événements

Il est possible d'attacher des listeners pour répondre aux événements liés à la sélection d'un élément ou l'extension ou la refermeture d'un noeud.

14.8.5.1. La classe TreePath

Durant son utilisation, le composant JTree ne gère pas directement les noeuds du modèle de données. La manipulation de ces noeuds se fait via un index ou une instance de la classe TreePath.

L'utilisation de l'index est assez délicate car seul le noeud racine de l'arbre possède toujours le même index 0. Pour les autres noeuds, la valeur de l'index dépend de l'état étendu/refermée de chaque noeud puisque seuls les noeuds affichés possèdent un index. Il est donc préférable d'utiliser la classe TreePath.

Le modèle de données utilise des noeuds mais l'interface de l'arbre utilise une autre représentation sous la forme de la classe TreePath.

La classe DefaultMutableTreeNode est la représentation physique d'un noeud, la classe TreePath est la représentation logique. Elle encapsule le chemin du noeud dans l'arborescence.

Cette classe contient plusieurs méthodes :

public Object getLastPathComponent();
public Object getPathComponent(int index);
public int getPathCount();
public Object[] getPath();
public TreePath getParentPath();
public TreePath pathByAddingChild(Object child);
public boolean isDescendant(TreePath treePath)

La méthode getPath() renvoie contenant chaque noeud qui compose le chemin encapsulé par la classe TreePath.

La méthode getLastPathComponent() renvoie le dernier noeud du chemin. Cette méthode permet d'obtenir à partir l'instance de la classe TreeNode correspondante dans le modèle de données.

La méthode getPathCount() renvoie le nombre de noeud qui compose le chemin.

La méthode getPathComponent() permet de renvoyer le noeud dont l'index dans le chemin est fourni en paramètre. L'élément avec l'index 0 est toujours le noeud racine de l'arbre.

La méthode getParentPath() renvoie une instance de la classe TreePath qui encapsule le chemin vers le noeud père du chemin encapsulé.

La méthode pathByAddingChild() renvoie une instance de la classe TreePath qui encapsule le chemin issu de l'ajout d'un noeud fils fourni en paramère.

La méthode idDescendant() renvoie un booléen qui précise si le chemin passé en paramère est un descendant du chemin encapsulé.

La classe TreePath ne permet pas de gérer le contenu de chaque noeud mais uniquement le chemin de ce noeud dans l'arboresence. Pour accéder au noeud à partir de son chemin, il faut utiliser la méthode getLastPathComponent(). Pour obtenir un noeud inclus dans le chemin, il faut utiliser la getPathComponent() ou getPath(). Toutes ces méthodes renvoient un objet ou un tableau de type Object. Il est donc nécessaire de réaliser un cast vers le type de noeud utilisé, généralement de type DefaultMutableTreeNode.

A partir d'un noeud de type DefaultMutableTreeNode, il est possible d'obtenir l'objet TreePath encapsulant le chemin du noeud en utilisant la méthode getPath() pour obtenir un tableau d'objets de type TreeNode et de passer ce tableau au constructeur de la classe TreePath.

Exemple (code Java 1.1) :
`TreeNode[] chemin = noeud.getPath(); TreePath path = new TreePath(chemin);`

14.8.5.2. La gestion de la sélection d'un noeud

La gestion de la selection de noeud dans un composant JTree est déléguée à un modèle de sélection sous la forme d'une classe qui implémente l'interface TreeSelectionModel. Par défaut, le composant JTree utilise une instance de la classe DefaultTreeSelectionModel.

Le modèle de sélection peut être configuré selon trois modes :

- SINGLE_TREE_SELECTION: un seul noeud peut être sélectionné.
- CONTIGUOUS_TREE_SELECTION: plusieurs noeuds peuvent être sélectionnés à condition d'être contigus.
- DISCONTIGUOUS_TREE_SELECTION: plusieurs noeuds peuvent être sélectionnés de façon continue et/ou discontinue (c'est le mode par défaut).

Pour empêcher la sélection d'un noeud dans l'arbre, il faut supprimer son modèle de sélection en passant null à la méthode setSelectionModel().

Exemple (code Java 1.1) :

```
JTree jTree = new JTree()jTree.setSelectionModel(null);
```

La sélection d'un noeud peut être réalisée par l'utilisateur ou par l'application : le modèle de sélection s'assure que celle−ci est réalisée en respectant le mode de sélection du modèle.

L'utilisateur peut utiliser la souris pour sélectionner un noeud ou appuyer sur la touche Espace sur le noeud courant pour le sélectionner. Il est possible de sélectionner plusieurs noeuds en fonction du mode en maintenant la touche CTRL enfoncée. Avec la touche SHIFT, il est possible selon le mode de sélectionner tous les noeuds entre un premier noeud sélectionné et le noeud sélectionné.

La sélection d'un noeud génère un événement de type TreeSelectionEvent.

Le dernier noeud sélectionné peut être obtenu en utilisant les méthodes getLeadSelectionPath() ou getLeadSelectionRow().

Par défaut la sélection d'un noeud entraine l'extension des noeuds ascendants correspondant afin de les rendre visibles. Pour empêcher ce comportement, il faut utiliser la méthode setExpandSelectedPath() en lui fournissant la valeur false en paramètre.

public void setExpandsSelectedPaths(boolean cond);

Les classes DefaultTreeSelectionModel et JTree possèdent plusieurs méthodes pour gérer la sélection de noeuds. Certaines de ces méthodes sont communes à ces deux classes.

Méthode	Rôle
int getSelectionMode()	renvoie le mode de sélection
void setSelectionMode(int mode)	mettre à jour le mode de sélection
Object getLastSelectedPathComponent()	renvoie le premier noeud de la sélection courante ou null si aucun noeud n'est sélectionné JTree uniquement
TreePath getAnchorSelectionPath()	JTree uniquement
voidsetAnchorSelectionPath(TreePathpath)	JTree uniquement
TreePath getLeadSelectionPath()	
setLeadSelectionPath()	

int getMaxSelectionRow()	Renvoie le plus grand index de la sélection
int getMinSelectionRow()	Renvoie le plus petit index de la sélection
int getSelectionCount()	Renvoie le nombre de noeud inclus dans la sélection
TreePath getSelectionPath()	Renvoie le chemin du premier élément sélectionné
TreePath[] getSelectionPaths()	Renvoie un tableau des chemins des noeuds inclus dans la sélection
int[] getSelectionRows()	Renvoie un tableau des index des noeuds inclus dans la sélection
Boolean isPathSelected (TreePath path)	Renvoie un booléen si le noeud dont le chemin est fourni en paramètre est inclus dans la sélection
Boolean isRowSelected(int row)	Renvoie un booléen si le noeud dont l'index est fourni en paramètre est inclus dans la sélection
boolean isSelectionEmpty()	Renvoie un booléen qui précise si la sélection est vide
void clearSelection()	Vide la sélection
void removeSelectionInterval (int row0, int row1)	Enlève de la sélection les noeuds dans l'intervalle des index fournis en paramètre
void removeSelectionPath(TreePath path)	Enlève de la sélection le noeud dont le chemin est fourni en paramètre
void removeSelectionRow (int row)	Enlève de la sélection le noeud dont l'index est fourni en paramètre JTree uniquement
void removeSelectionRows(int[] rows)	Enlève de la sélection les noeuds dont les index sont fournis en paramètre JTree uniquement
void addSelectionInterval(int row0, int row1)	Ajouter à la selection les noeuds dont l'intervalle des index est fourni en paramètre
void addSelectionPath(TreePath path)	Ajouter à la sélection le noeud dont le chemin est fourni en paramètre
addSelectionPaths(TreePath[] path)	Ajouter à la sélection les noeuds dont les chemins sont fournis en paramètre
void addSelectionRow(int row)	Ajouter à la sélection le noeud dont l'index est fourni en paramètre
void addSelectionRows(int[] row)	Ajouter à la sélection les noeuds dont les index sont fournis en paramètre
void setSelectionInterval(int row0, int row1)	Définir la sélection avec les noeuds dont les index sont fournis en paramètre JTree uniquement
setSelectionPath(TreePath path)	Définir la sélection avec le noeud dont le chemin est fourni en paramètre
void setSelectionPaths (TreePath[] path)	Définir la sélection avec les noeuds dont les chemins sont fournis en paramètre
void setSelectionRow(int row)	Définir la sélection avec le noeud dont l'index est fourni en paramètre
void setSelectionRows(int[] row)	Définir la sélection avec les noeuds dont les index sont fournis en paramètre JTree uniquement

14.8.5.3. Les événements lies à la selection de noeuds

Lors de la sélection d'un noeud, un événement de type TreeSelectionEvent est émis. Pour traiter cet événement, le composant doit enregistrer un listener de type TreeSelectionListener.

L'interface TreeSelectionListener définit une seule méthode :

public void valueChanged(TreeSelectionEvent evt)

Exemple (code Java 1.1) :

```
jTree.addTreeSelectionListener(new javax.swing.event.TreeSelectionListener() {
      public void valueChanged(javax.swing.event.TreeSelectionEvent e) {

        DefaultMutableTreeNode noeud = (DefaultMutableTreeNode) jTree
            .getLastSelectedPathComponent();
        if (noeud == null)
          return;
        System.out.println("valueChanged() : " + noeud);
      }
});
```

La classe TreeSelectionEvent possède plusieurs méthodes pour obtenir des inforamtions sur la sélection.

Méthode	Rôle
public TreePath[] getPaths()	renvoie un tableau des chemins des noeuds sélectionnés
public boolean isAddedPath (TreePath path)	Renvoie true si le noeud sélectionné est ajouté à la sélection. Renvoie false si le noeud sélectionné est retiré de la selection
TreePath getPath()	Renvoie le chemin du premier noeud sélectionné
boolean isAddedPath()	Renvoie true si le premier noeud sélectionné est ajouté à la sélection. Renvoie false si le premier noeud sélectionné est retiré de la selection
TreePath getOldLeadSelection()	
TreePath getNewLeadSelection()	

Un listener de type TreeSelectionListener est engistré en utilisant la méthode addTreeSelectionListener()de la classe JTree.

Exemple (code Java 1.1) :

```
jTree.addTreeSelectionListener(new TreeSelectionListener() {

  public void valueChanged(TreeSelectionEvent e) {

    Object obj = jTree.getLastSelectedPathComponent();
    System.out.println("getLastSelectedPathComponent=" + obj);
    System.out.println("getPath=" + e.getPath());
    System.out.println("getNewLeadSelectionPath="
        + e.getNewLeadSelectionPath());
    System.out.println("getOldLeadSelectionPath="
        + e.getOldLeadSelectionPath());
    TreePath[] paths = e.getPaths();

    for (int i = 0; i < paths.length; i++) {
      System.out.println("Path " + i + "=" + paths[i]);
    }
  }
});
```

Un événement de type TreeSelectionEvent n'est émis que si un changement intervient dans la sélection : lors d'un clic sur un noeud, celui est sélectionné et un événement est émis. Lors d'un clic sur ce même noeud, le noeud est toujours sélectionné mais l'événement n'est pas émis puisque la sélection n'est pas modifiée.

Dans un listener pour gérer les événements de la souris, il est possible d'utiliser la méthode getPathForLocation() pour déterminer le chemin d'un noeud à partir des coordonnées de la souris qu'il faut lui fournir en paramètre.

La méthode getPathForLocation() renvoie null si l'utilisateur clic en dehors d'un noeud dans l'arbre.

Exemple (code Java 1.1) :

```
jTree.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent evt) {
      TreePath path =
         jTree.getPathForLocation(evt.getX(), evt.getY());
      if (path != null) {
         System.out.println("path= " + path.getLastPathComponent());
       }
  }
});
```

Plusieurs autres méthodes peuvent aussi être utilisées dans ce contexte.

Méthode	Rôle
TreePath getClosestPathForLocation(int x, int y)	Retourne le chemin du noeud le plus proche des coordonnées fournies en paramètre
int getClosestRow ForLocation(int x, int y)	Retourne l'index du noeud le plus proche des coordonnées fournies en paramètre
Rectangle getPathBounds(TreePath path)	Renvoie un objet de type Rectangle qui représente la surface du noeud dont le chemin est fourni en paramètre
TreePath getPathForLocation(int x, int y)	Retourne le chemin du noeud dont la surface contient les coordonnées fournies en paramètre. Renvoie null si ces coorodonnées ne correspondent à aucune
TreePath getPathForRow(int row)	Renvoie le chemin du noeud dont l'index est fourni en paramètre
Rectangle getRow−Bounds(int row)	Renvoie un objet de type Rectangle qui représente la surface du noeud dont l'index est fourni en paramètre
int getRowForLocation(int x, int y)	

14.8.5.4. Les événements lorsqu'un noeud est étendu ou refermé

A chaque fois qu'un noeud est étendu ou refermé, un événement de type TreeExpansionEvent est émis. Il est possible de répondre à ces événements en mettant en place un listener de type TreeExpansionListener.

L'interface TreeExpansionListener propose deux méthodes :

public void treeExpanded(TreeExpansionEvent event) public void treeCollapsed(TreeExpansionEvent event)

La classe TreeExpansionEvent possède une propriété source qui contient une référence sur le composant JTree à l'origine de l'événement et une propriété path qui contient un objet de type TreePath encapsulant le chemin du noeud à l'origine de l'événement.

Les valeurs de ces deux propriétés peuvent être obtenus avec leur getter respectif : getSource() et getPath().

Exemple (code Java 1.1) :

```
jTree.addTreeExpansionListener(new TreeExpansionListener() {
```

```
  public void treeExpanded(TreeExpansionEvent evt) {
    System.out.println("expand, path=" +
                         evt.getPath());
  }

  public void treeCollapsed(TreeExpansionEvent evt) {
    System.out.println("collapse, path=" +
                       evt.getPath());
  }

});
```

Un seul événement est généré à chaque fois qu'un noeud est étendu ou refermé : il n'y a pas d'événements émis pour les éventuels noeuds fils qui sont étendu ou refermé suite à l'action.

14.8.5.5. Le contrôle des actions pour étendre ou refermer un noeud

Il peut être utile de recevoir un événement avant qu'un noeud ne soit étendu ou refermé. Un listener de type TreeWillExpandListener() peut être mis en place pour recevoir un événement de type TreeExpansionEvent lors d'une tentative pour étendre ou refermé un noeud.

L'interface TreeWillExpandListener définit deux méthodes :

public void treeWillCollapse(TreeExpansionEvent evt) throws ExpandVetoException;
public void treeWillExpand(TreeExpansionEvent evt) throws ExpandVetoException;

Les deux methodes peuvent lever une exception de type ExpandVetoException. Cette méthode doit lever cette exception dans les traitements de ces méthodes si des conditions sont remplies pour empecher l'action demandée par l'utilisateur. Si l'exception n'est pas levée à la fin des traitements de la méthode alors l'action est réalisée.

Exemple (code Java 1.1) : empécher tous les noeuds étendus de se refermer

```
jTree.addTreeWillExpandListener(new TreeWillExpandListener() {

  public void treeWillCollapse(TreeExpansionEvent event) throws ExpandVetoException {
    throw new ExpandVetoException(event);
  }

  public void treeWillExpand(TreeExpansionEvent event) throws ExpandVetoException {
  }

});
```

14.8.6. La personnalisation du rendu

Le rendu du composant JTree dépend bien sûre dans un premier temps du look and feel utilisé mais il est aussi possible de personnaliser plus finement le rendu des noeuds du composant.

Il est possible de préciser la façon dont les lignes reliant les noeuds sont rendues via une propriété client nommée lineStyle. Cette propriété peut prendre trois valeurs :

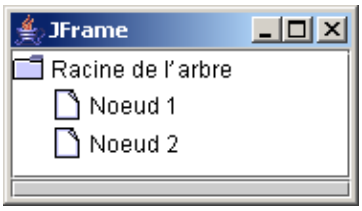
Valeur	Rôle
Angled	Une ligne à angle droit relie chaque noeud fils à son noeud père
None	Aucune ligne n'est affichée entre les noeuds
Horizontal	Une simple ligne horizontale sépare les noeuds enfants du noeud racine

Pour préciser la valeur de la propriété que le composant doit utiliser, il faut utiliser la méthode putClientProperty() qui attend deux paramètres sous forme de chaînes de caractères :

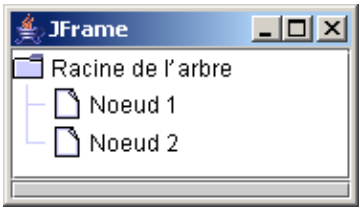
- le nom de la propriété
- sa valeur

Exemple (code Java 1.1) :

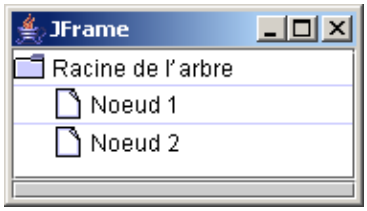
```
jTree = new JTree(racine);
jTree.putClientProperty("JTree.lineStyle","Horizontal");
```



None



Angled



Horizontal

Il est possible de modifier l'apparence de la racine de l'arbre grace à deux méthodes de la classe JTree : setRootVisible() et setShowsRootHandles().

La méthode setRootVisible() permet de préciser avec son booléen en paramètre si la racine est affichée ou non.

Exemple (code Java 1.1) :

```
JTree jtree = new JTree();
jtree.setShowsRootHandles(false);
jtree.setRootVisible(true);
```

14.8.6.1. Personnaliser le rendu des noeuds

Il est possible d'obtenir un contrôle total sur le rendu de chaque noeud en définissant un objet qui implémente l'interface TreeCellRender. Attention, le rendu personnalisé est parfois dépendant du look & feel utilisé.

L'interface TreeCellRenderer ne définit qu'une seule méthode :

Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)

Cette méthode envoie un composant qui va encapsuler le rendu du noeud. Le premier argument de type JTree encapsule le composant JTree lui−même.L'argument de type Object encapsule le noeud dont le rendu doit être généré.

La méthode getCellRender() renvoie un objet qui encapsule le TreeCellRender. Il est nécessaire de réaliser un cast vers le type de cet objet.

Swing propose une classe de base DefaultTreeCellRender pour le rendu. Elle propose plusieurs méthodes pour permettre de définir le rendu.

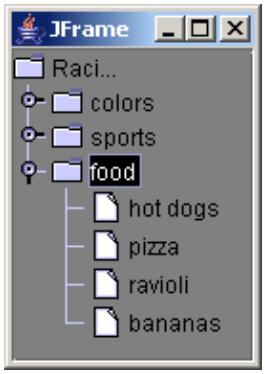
Méthode	Rôle
void setBackgroundNonSelectionColor(Color)	Permet de définir la couleur de fond du noeud lorsqu'il n'est pas sélectionné
void setBackgroundSelectionColor(Color)	Permet de définir la couleur de fond du noeud lorsqu'il est sélectionné
void setBorderSelectionColor(Color)	Permet de définir la couleur de la bordure du noeud lorsqu'il est sélectionné. Il n'est pas possible de définir une bordure pour un noeud sélectionné
void setTextNonSelectionColor(Color)	Permet de définir la couleur du texte du noeud lorsqu'il n'est pas sélectionné
void setTextSelectionColor(Color)	Permet de définir la couleur du texte du noeud lorsqu'il est sélectionné
void setFont(Font)	Permet de définir la police de caractère utilisé pour afficher le texte du noeud
void setClosedIcon(Icon)	Permet de définir l'icône associée au noeud lorsque celui−ci est fermé
void setOpenIcon(Icon)	Permet de définir l'icône associée au noeud lorsque celui−ci est étendu
void setLeafIcon(Icon)	Permet de définir l'icône associée au noeud lorsque celui−ci est une feuille

Un composant ne peut avoir qu'une seule instance de type TreeCellRenderer. Cette instance sera donc appelée pour définir le rendu de chaque noeud.

Exemple (code Java 1.1) :

```
TreeCellRenderer cellRenderer = jTree.getCellRenderer();
if (cellRenderer instanceof DefaultTreeCellRenderer) {
  DefaultTreeCellRenderer renderer = (DefaultTreeCellRenderer)cellRenderer;
  renderer.setBackgroundNonSelectionColor(Color.gray);
  renderer.setBackgroundSelectionColor(Color.black);
  renderer.setTextSelectionColor(Color.white);
  renderer.setTextNonSelectionColor(Color.black);
  jTree.setBackground(Color.gray);
}
```

Résultat :



Pour modifier les icônes utiliser par les différents élément de l'arbre, il faut utiliser les méthodes setOpenIcon(), setClosedIcon() et setLeafIcon().

Méthode	Rôle

setOpenIcon()	précise l'icône pour un noeud ouvert
setClosedIcon()	précise l'icône pour un noeud fermé
setLeafIcon()	précise l'icône pour une feuille

Pour simplement supprimer l'affichage de l'icône, il suffit de passer null à la méthode concernée ou de lui fournir une image sous la forme d'un objet de type

Exemple (code Java 1.1) :

```
DefaultTreeCellRenderer monRenderer = new DefaultTreeCellRenderer();
monRenderer.setOpenIcon(null);
monRenderer.setClosedIcon(null);
monRenderer.setLeafIcon(null);
```

Pour préciser une image, il est faut créer une instance de la classe ImageIcon encapsulant l'image et la passer en paramètre de la méthode concernée.

Exemple (code Java 1.1) :

```
    private Icon ourvertIcon = new ImageIcon("images/ouvert.gif");
    private Icon fermeIcon   = new ImageIcon("images/ferme.gif");
    private Icon feuilleIcon = new ImageIcon("images/feuille.gif");
...
    DefaultTreeCellRenderer treeCellRenderer = new DefaultTreeCellRenderer();
    treeCellRenderer.setOpenIcon(ouvertIcon);
    treeCellRenderer.setClosedIcon(fermeIcon);
    treeCellRenderer.setLeafIcon(feuilleIcon);
```

Il est aussi possible de définir une classe qui hérite de la classe DefaultTreeCellRenderer. Cette classe propose une implémentation par défaut de l'interface TreeCellRenderer. Comme elle hérite de la classe JLabel, elle possède déjà de nombreuses méthodes pour assurer le rendu du noeud sous la forme d'un composant de type étiquette.

Exemple (code Java 1.1) :

```
import java.awt.Color;
import java.awt.Component;

import javax.swing.JTree;
import javax.swing.tree.DefaultTreeCellRenderer;

public class MonTreeCellRenderer extends DefaultTreeCellRenderer {

  public Component getTreeCellRendererComponent(JTree tree, Object value,
      boolean selected, boolean expanded, boolean leaf, int row,
      boolean hasFocus) {


    super.getTreeCellRendererComponent(tree,value, selected, expanded,
        leaf, row,hasFocus);

    setBackgroundNonSelectionColor(Color.gray);
    setBackgroundSelectionColor(Color.black);
    setTextSelectionColor(Color.white);
    setTextNonSelectionColor(Color.black);

    return this;
  }
}
```

Une fois la classe de type DefaultTreeCellRenderer instanciée, il faut utiliser la méthode setCellRenderer() de la classe JTree pour indiquer à l'arbre d'utiliser cette classe pour le rendu.

Exemple (code Java 1.1) :

```
    jTree.setCellRenderer(new MonTreeCellRenderer());
```

La création d'une classe fille de la classe DefaultTreeCellRenderer ne fonctionne correctement qu'avec les look and feel Metal et Windows car le look and feel Motif définit son propre Renderer.

14.8.6.2. Les bulles d'aides (Tooltips)

Le composant JTree ne propose par de support pour les bulles d'aide en standard. Pour permettre à un composant JTree d'afficher une bulle d'aide, il faut :

- enregistrer le composant JTree auprès du ToolTipManager
- définir le contenu de la bulle d'aide dans le Renderer

L'enregistrement du composant auprès du ToolTipManager se fait en utilisant la méthode registerComponent() sur l'instance partagée.

Exemple (code Java 1.1) :

```
ToolTipManager.sharedInstance().registerComponent(jTree);
((JLabel)t.getCellRenderer()).setToolTipText("Arborescence des données");
```

L'inconvénient de cette méthode est que la bulle d'aide est toujours la même quelque soit la position de la souris sur tous les noeuds du composant. Pour assigner une bulle d'aide particulière à chaque noeud, il est nécessaire d'utiliser la méthode setToolTipText() dans la méthode getTreeCellRendererComponent() d'une instance fille de la classe DefaultTreeCellRenderer

Exemple (code Java 1.1) :

```
      jTree.setCellRenderer(new DefaultTreeCellRenderer() {

        public Component getTreeCellRendererComponent(JTree tree, Object value,
            boolean selected, boolean expanded, boolean leaf, int row,
            boolean hasFocus) {

          super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf, row,
              hasFocus);
          setToolTipText(value.toString());

          return this;
        }
      });

      ToolTipManager.sharedInstance().registerComponent(jTree);
```

La suite de ce chapitre sera développée dans une version future de ce document

15. Le développement d'interfaces graphiques avec SWT

Ce chapitre contient plusieurs sections :

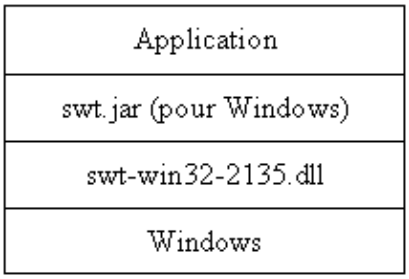
15.1. Présentation

La première API pour développer des interfaces graphiques portables d'un système à un autre en Java est AWT. Cette API repose sur les composants graphiques du système sous jacent ce qui lui assure de bonne performance. Malheureusement, ces composants sont limités dans leur fonctionnalité car ils représentent le petit dénominateur commun aux communs des différents systèmes concernés.

Pour palier à ce problème, Sun a proposé une nouvelle API, Swing. Cette Api est presque exclusivement écrite en Java, ce qui assure sa portabilité. Swing possède aussi d'autres points forts, tel que des fonctionnalités avancées, la possibilité d'étendre les composants, une adaptation du rendu de composants, etc ... Swing est une API mature, éprouvée et parfaitement connue. Malheureusement, ces deux gros défauts sont sa consommation en ressource machine et la lenteur d'exécution des applications qui l'utilise.

SWT propose une approche intermédiaire : utiliser autant que possible les composants du système et implémenter les autres composants en Java. SWT est écrit en Java et utilise la technologie JNI pour appeler les composants natifs. SWT utilise autant que possible les composants natifs du système lorsque ceux sont présentés, sinon ils sont réécrit en pur Java. Les données de chaque composant sont aussi stockées autant que possible dans le composant natif, limitant ainsi les données stockées dans les objets Java correspondant.

Ainsi, une partie de SWT est livrée sous la forme d'une bibliothèque dépendante du système d'exploitation et d'un fichier .jar lui aussi dépendant du système. Toutes les fonctionnalités de SWT ne sont implémentées que sur les systèmes ou elles sont supportées (exemple, l'utilisation des ActiveX n'est possible que sur le portage de SWT sur les systèmes Windows).



Les trois avantages de SWT sont donc la rapidité d'exécution, des ressources machines moins importantes lors de l'exécution et un rendu parfait des composants graphiques selon le système utilisé puisqu'il utilise des composants natifs. Cette dernière remarque est particulièrement vraie pour des environnements graphiques dont l'apparence est modifiable.

Malgré cette dépendance vis à vis du système graphique de l'environnement d'exécution, l'API de SWT reste la même quelque soit la plate−forme utilisée.

En plus de dépendre du système utilisé lors de l'exécution, SWT possède un autre petit inconvénient. N'utilisant pas de purs objets java, il n'est pas possible de compter sur le ramasse miette pour libérer la mémoire des composants créés manuellement. Pour libérer cette mémoire, il est nécessaire d'utiliser la méthode dispose() pour les composants instanciés lorsque ceux ci ne sont plus utiles.

Pour faciliter ces traitements, l'appel de la méthode dispose() d'un composant entraîne automatiquement l'appel de la méthode dispose() des composants qui lui sont rattachés. Il faut toutefois rester vigilant lors de l'utilisation de certains objets qui ne sont pas des contrôles tel que les objets de type Font ou Color, qu'il convient de libérer explicitement sous peine de fuites de mémoire.

Les règles à observer pour la libération des ressources sont :

- toujours appeler la méthode dispose() de tout objet non rattaché directement à un autre objet qui n'est plus utilisé.
- ne jamais appeler la méthode dispose() d'objets qui n'ont pas été explicitement instanciés dans le code
- l'appel de la méthode dispose() d'un composant entraîne automatiquement l'appel de la méthode dispose() des composants qui lui sont rattachés

Attention, l'utilisation d'un objet dont la méthode dispose() a été appelée induira un résultat imprévisible.

Ainsi SWT repose la problématique concernant la dualité entre la portabilité (Write Once Run Anywhere) et les performances.

SWT repose sur trois concepts classiques dans le développement d'une interface graphique :

- Les composants ou contrôles (widgets)
- Un système de mise en page et de présentation des composants
- Un modèle des gestions des événements

La structure d'une application SWT est la suivante :

- la création d'un objet de type Display qui assure le dialogue avec le système sous jacent
- la création d'un objet de type Shell qui est la fenêtre de l'application
- la création des composants et leur ajout dans le Shell
- l'enregistrements des listeners pour le traitement des événements
- l'exécution de la boucle de gestion des événements jusqu'à la fin de l'application
- la libération des ressources de l'objet Display

La version de SWT utilisée dans ce chapitre est la 2.1.

SWT est regroupé dans plusieurs packages :

Package	Rôle
org.eclipse.swt	Package de base qui contient la définition de constantes et d'exceptions

org.eclipse.swt.accessibility	
org.eclipse.swt.custom	Contient des composants particuliers
org.eclipse.swt.dnd	Contient les éléments pour le support du « cliqué / glissé »
org.eclipse.swt.events	Contient les éléments pour la gestion des événements
org.eclipse.swt.graphics	Contient les éléments pour l'utilisation des éléments graphiques (couleur, polices, curseur, contexte graphique, ...)
org.eclipse.swt.layout	Contient les éléments pour la gestion de la présentation
org.eclipse.swt.ole.win32	Contient les éléments pour le support de OLE 32 sous Windows
org.eclipse.swt.printing	Contient les éléments pour le support des impressions
org.eclipse.swt.program	
org.eclipse.swt.widgets	Contient les différents composants

15.2. Un exemple très simple

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class TestSWT1 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);

    Label label = new Label(shell, SWT.CENTER);
    label.setText("Bonjour!");
    label.pack();

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
    label.dispose();
  }
}
```

Pour exécuter cet exemple sous windows, il faut que le fichier swt.jar correspondant à la plate−forme Windows soit inclus dans le classpath et que l'application puisse accéder à la bibliothèque swt−win32−2135.dll.

15.3. La classe SWT

Cette classe définit un certain nombre de constante concernant les styles. Les styles sont des comportement ou des

caractéristiques définissant l'apparence du composant. Ces styles sont directement fournis dans le constructeur d'une classe encapsulant un composant.

15.4. L'objet Display

Toute application SWT doit obligatoirement instancier un objet de type Display. Cet objet assure le dialogue entre l'application et le système graphique du système d'exploitation utilisé.

```
Display display = new Display();
```

La méthode la plus importante de la classe Display est la méthode readAndDispatch() qui lit les événements dans la pile du système graphique natif pour les diffuser à l'application. Elle renvoie true si il y a encore des traitements a effectuer sinon elle renvoie false.

La méthode sleep() permet de mettre en attente le thread d'écoute de événements jusqu'à l'arrivée d'un nouvel événement.

Il est absolument nécessaire lors de la fin de l'application de libérer les ressources allouées par l'objet de type Display en appelant sa méthode dispose().

15.5. L'objet Shell

L'objet Shell représente une fenêtre gérée par le système graphique du système d'exploitation utilisé.

Un objet de type Shell peut être associé à un objet de type Display pour obtenir une fenêtre principale ou être associé à un autre objet de type Shell pour obtenir une fenêtre secondaire.

La classe Shell peut utiliser plusieurs styles : BORDER,H_SCROLL, V_SCROLL, CLOSE, MIN, MAX, RESIZE, TITLE, SHELL_TRIM, DIALOG_TRIM

BORDER : une fenêtre avec une bordure sans barre de titre
 	
`Shell shell = new Shell(display, SWT.BORDER);`
`shell.setSize(200, 100) ;`
`shell.setText("test");`
	

TITRE : une fenêtre avec une barre de titre
 	
`Shell shell = new Shell(display, SWT.BORDER);`
`shell.setSize(200, 100) ;`
`shell.setText("test");`
	

CLOSE : une fenêtre avec un bouton de fermeture
 	
`Shell shell = new Shell(display, SWT.BORDER | SWT.CLOSE);`
`shell.setSize(200, 100);`
`shell.setText("test");`
	

MIN : une fenêtre avec un bouton pour iconiser
 	
`Shell shell = new Shell(display, SWT.BORDER | SWT.CLOSE |`
`  SWT.MIN);`
`shell.setSize(200, 100);`
`shell.setText("test");`
	

MAX : une fenêtre avec un bouton pour agrandir au maximum
 	
`Shell shell = new Shell(display, SWT.BORDER | SWT.CLOSE | SWT.MAX);`
`shell.setSize(200, 100);`
`shell.setText("test");`
	

RESIZE : une fenêtre dont la taille peut être modifiée
 	
`Shell shell = new Shell(display, SWT.CLOSE | SWT.RESIZE);`
`shell.setSize(200, 100);`
`shell.setText("test");`
	

SHELL_TRIM : groupe en une seule constante les style CLOSE, TITLE, MIN, MAX et RESIZE	
DIALOG_TRIM : groupe en une seule constante les style CLOSE, TITLE et BORDER	
APPLICATION_MODAL :	
SYSTEM_MODAL :	

La méthode setSize() permet de préciser la taille de la fenêtre.

La méthode setTexte() permet de préciser le titre de la fenêtre.

Exemple : centrer la fenêtre sur l'écran

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.*;

public class TestSWT1 {

  public static void centrerSurEcran(Display display, Shell shell) {
    Rectangle rect = display.getClientArea();
    Point size = shell.getSize();
    int x = (rect.width - size.x) / 2;
    int y = (rect.height - size.y) / 2;
    shell.setLocation(new Point(x, y));
  }

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setSize(340, 100);
    centrerSurEcran(display, shell);

    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
```

```
}
```

15.6. Les composants

Les composants peuvent être regroupés en deux grandes familles :

- les contrôles qui sont des composants graphiques. Ils héritent tous de la classe abstraite Control
- les conteneurs qui permettent de grouper des contrôles. Ils héritent tous de la classe abstraite Composite

Un application SWT est une hiérarchie de composants dont la racine est un objet de type Shell.

Certaines caractéristiques comme l'apparence ou le comportement d'un contrôle doivent être fournies au moment de leur création par le système graphique. Ainsi, chaque composant SWT possède une propriété nommée style fournie en paramètre du constructeur.

Plusieurs styles peuvent être combinés avec l'opérateur | . Cependant certains styles sont incompatibles entre eux pour certains composants.

15.6.1. La classe Control

La classe Control définit trois styles : BORDER, LEFT_TO_RIGHT et RIGHT_TO_LEFT

Le seul constructeur de la classe Control nécessite aussi de préciser le composant père sous la forme d'un objet de type Composite. L'association avec le composant père est obligatoire pour tous les composants lors de leur création.

La classe Control possède plusieurs méthodes pour enregistrer des listeners pour certains événements. Ces événements sont : FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize.

Elle possède aussi plusieurs méthodes dont les principales sont :

Nom	Rôle
boolean forceFocus()	Force le focus au composant pour lui permettre de recevoir les événements clavier
Display getDisplay()	Renvoie l'objet Display associé au composant
Shell getShell()	Renvoie l'objet Shell associé au composant
void pack()	Recalcule la taille préférée du composant
void SetEnabled()	Permet de rendre actif le composant
void SetFocus()	Donne le focus au composant pour lui permettre de recevoir les événements clavier
void setSize()	Permet de modifier la taille du composant
void setVisible()	Permet de rendre visible ou non le composant

15.6.2. Les contrôles de base

15.6.2.1. La classe Button

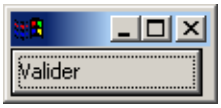
La classe Button représente un bouton cliquable.

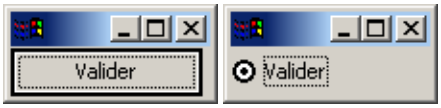
La classe Button définit plusieurs styles : BORDER, CHECK, PUSH, RADIO, TOGGLE, FLAT, LEFT, RIGHT, CENTER, ARROW (avec UP, DOWN)

NONE : un bouton par défaut
 	
`button = new Button(shell, SWT.NONE);`
`button.setText("Valider");`
`button.setSize(100,25);`
	

BORDER : met une bordure autour du bouton `Button button = new Button(shell, SWT.BORDER);`	
CHECK : une case à cocher `Button button = new Button(shell, SWT.CHECK);`	
RADIO : un bouton radio `Button button = new Button(shell, SWT.RADIO);`	
PUSH : un bouton standard (valeur par défaut) `Button button = new Button(shell, SWT.PUSH);`	
TOGGLE : un bouton pouvant concervé un état enfoncé `Button button = new Button(shell, SWT.TOGGLE);`	

ARROW : bouton en forme de flèche (par défaut vers le haut)
 	
`Button button = new Button(shell, SWT.ARROW);`
`Button button = new Button(shell, SWT.ARROW | SWT.DOWN);`
	

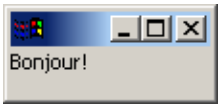
RIGHT : position le contenu du bouton vers la droite `Button button = new Button(shell, SWT.RIGHT);`	
LEFT : position le contenu du bouton vers la gauche `Button button = new Button(shell, SWT.LEFT);`	
CENTER : position le contenu du bouton au milieu `Button button = new Button(shell, SWT.CENTER);`	

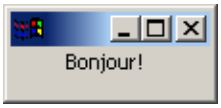
FLAT : le bouton apparaît en 2D
 	
`Button button = new Button(shell, SWT.FLAT);`
`Button button = new Button(shell, SWT.FLAT | SWT.RADIO);`
	

15.6.2.2. La classe Label

Ce contrôle permet d'afficher un libellé ou une image

La classe Label possède plusieurs styles : BORDER, CENTER, LEFT, RIGHT, WRAP, SEPARATOR (avec HORIZONTAL, SHADOW_IN, SHADOW_OUT, SHADOW_NONE, VERTICAL)

NONE : un libellé par défaut
 	
`Label label = new Label(shell, SWT.NONE);`
`label.setText("Bonjour!");`
`label.setSize(100,25);`
	

BORDER : ajouter une bordure autour du libellé `Label label = new Label(shell, SWT.BORDER);`	
CENTER : permet de centré le libellé `Label label = new Label(shell, SWT.CENTER);`	
SEPARATOR et VERTICAL : une barre verticale `Label label = new Label(shell, SWT.SEPARATOR | SWT. VERTICAL);`	
SEPARATOR et HORIZONTAL : une barre horizontale `Label label = new Label(shell, SWT.SEPARATOR | SWT.HORIZONTAL);`	
SHADOW_IN : `Label label = new Label(shell, SWT.SEPARATOR | SWT.HORIZONTAL | SWT.SHADOW_IN);`	
SHADOW_OUT : `Label label = new Label(shell, SWT.SEPARATOR | SWT.HORIZONTAL | SWT.SHADOW_OUT);`	

Cette classe possède plusieurs méthodes dont les principales sont :

Nom	Rôle
void setAlignment(int)	Permet de préciser l'alignement des données du contrôle
void setImage(Image)	Permet de préciser une image affichée par le contrôle
void setText(string)	Permet de préciser le texte du contrôle

15.6.2.3. La classe Text

Ce contrôle est une zone de saisie de texte.

La classe Text possède plusieurs styles : BORDER, SINGLE, READ_ONLY, LEFT, CENTER, RIGHT, WRAP, MULTI (avec H_SCROLL, V_SCROLL)

NONE : une zone de saisie sans bordure
 	
`Text text = new Text(shell, SWT.NONE);`
`text.setText("mon texte");`
`text.setSize(100,25);`
	

BORDER : une zone de saisie avec bordure `Text text = new Text(shell, SWT.BORDURE);`	
MULTI, SWT.H_SCROLL, SWT.V_SCROLL : une zone de saisie avec bordure `Text text = new Text(shell, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);`	
READ_ONLY : une zone de saisie en lecture seule	

Cette classe possède plusieurs méthodes dont les principales sont :

Nom	Rôle
void setEchoChar(char)	Caractère affiché lors de la frappe d'une touche
void setTextLimit(int)	Permet de préciser le nombre maximum de caractères saisissable
void setText(string)	Permet de préciser le contenu de la zone de texte
void setEditable(boolean)	Permet de rendre le contrôle éditable ou non

15.6.3. Les contrôles de type liste

SWT permet de créer de type liste et liste déroulante.

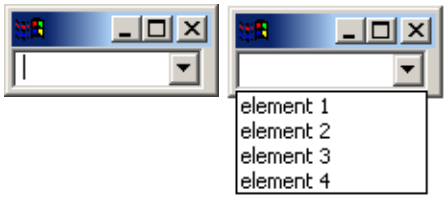
15.6.3.1. La classe Combo

Ce contrôle est une liste déroulante dans laquelle l'utilisateur peut sélectionner une valeur dans une liste d'éléments prédéfinis ou saisir un élément.

La classe Combo définit trois styles : BORDER, DROP_DOWN, READ_ONLY, SIMPLE

BORDER : une liste déroulante

```
Combo combo = new Combo(shell, SWT.BORDER);
combo.add("element 1");
combo.add("element 2");
combo.add("element 3");
combo.add("element 4");
combo.setSize(100,41);
```

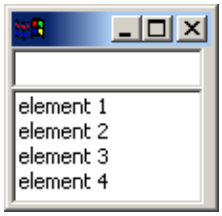


READ_ONLY : une liste déroulante ne permettant que la sélection (saisie d'un élément impossible)

```
Combo combo = new Combo(shell, SWT.BORDER | SWT.READ_ONLY);
```

SIMPLE : zone de saisie et une liste

```
Combo combo = new Combo(shell, SWT.BORDER | SWT.SIMPLE);
combo.setSize(100,81);
```



15.6.3.2. La classe List

Ce contrôle est une liste qui permet de sélectionner un ou plusieurs éléments.

La classe List possède plusieurs styles : BORDER, H_SCROLL, V_SCROLL, SINGLE, MULTI

BORDER : une liste

```
List liste = new List(shell, SWT.BORDER);
liste.add("element 1");
liste.add("element 2");
liste.pack();
```

V_SCROLL : une liste avec une barre de défilement
 	
`List liste = new List(shell, SWT.V_SCROLL);`
`liste.add("element 1");`
`liste.add("element 2");`
`liste.add("element 3");`
`liste.add("element 4");`
`liste.setSize(100,41);`
	

MULTI : une liste avec sélection de plusieurs éléments `List liste = new List(shell, SWT.V_SCROLL | SWT.MULTI);`	

La méthode add() permet d'ajouter un élément à la liste sous la forme d'un chaîne de caractères.

La méthode setItems() permet de fournir les éléments de la liste sous la forme d'un tableau de chaîne de caractères.

Exemple :

`List liste = new List(shell, SWT.V_SCROLL | SWT.MULTI);`
`liste.setItems(new String[] {"element 1", "element 2", "element 3", "element 4"});`
`liste.setSize(100,41);`

15.6.4. 1.6.4 Les contrôles pour les menus

SWT permet la création de menus principaux et de menus déroulants. La création de ces menus met en oeuvre deux classes : Menu, MenuItem

15.6.4.1. La classe Menu

Ce contrôle est un élément du menu qui va contenir des options

La classe Menu possède plusieurs styles : BAR, DROP_DOWN, NO_RADIO_GROUP, POP_UP

BAR : le menu principal d'une fenêtre
 	
`Menu menu = new Menu(shell, SWT.BAR);`
`MenuItem menuitem1 = new MenuItem(menu, SWT.CASCADE);`
`menuitem1.setText("Fichier");`
`shell.setMenuBar(menu);`
	

POP_UP : un menu contextuel
 	
`Menu menu = new Menu(shell, SWT.POP_UP);`
`MenuItem menuitem1 = new MenuItem(menu, SWT.CASCADE);`
`menuitem1.setText("Fichier");`
`MenuItem menuitem2 = new MenuItem(menu, SWT.CASCADE);`
`menuitem2.setText("Aide");`
`shell.setMenu(menu);`
	

DROP_DOWN : un sous menu	
NO_RADIO_GROUP :	

15.6.4.2. La classe MenuItem

Ce contrôle est une option d'un menu.

La classe MenuItem possède styles : CHECK, CASCADE, PUSH, RADIO, SEPARATOR

CASCADE : une option de menu qui possède un sous menu

PUSH : une option de menu

```
Menu menu = new Menu(shell, SWT.BAR);
MenuItem optionFichier = new MenuItem(menu, SWT.CASCADE);
optionFichier.setText("Fichier");
Menu menuFichier = new Menu(shell, SWT.DROP_DOWN);
MenuItem optionOuvrir = new MenuItem(menuFichier, SWT.PUSH);
optionOuvrir.setText("Ouvrir");
MenuItem optionFermer = new MenuItem(menuFichier, SWT.PUSH);
optionFermer.setText("Fermer");
optionFichier.setMenu(menuFichier);
MenuItem optionAide = new MenuItem(menu, SWT.CASCADE);
optionAide.setText("Aide");
shell.setMenuBar(menu);
```

CHECH : une option de menu avec un état coché ou non

```
Menu menu = new Menu(shell, SWT.BAR);
MenuItem optionFichier = new MenuItem(menu, SWT.CASCADE);
optionFichier.setText("Fichier");
Menu menuFichier = new Menu(shell, SWT.DROP_DOWN);
MenuItem optionOuvrir = new MenuItem(menuFichier, SWT.CASCADE);
optionOuvrir.setText("Ouvrir");
MenuItem optionFermer = new MenuItem(menuFichier, SWT.CASCADE);
optionFermer.setText("Fermer");
MenuItem optionCheck = new MenuItem(menuFichier, SWT.CHECK);
optionCheck.setText("Check");
optionFichier.setMenu(menuFichier);
MenuItem optionAide = new MenuItem(menu, SWT.CASCADE);
optionAide.setText("Aide");
shell.setMenuBar(menu);
```

Radio : un option de menu sélectionnable parmi un ensemble

```
Menu menu = new Menu(shell, SWT.BAR);
MenuItem optionFichier = new MenuItem(menu, SWT.CASCADE);
optionFichier.setText("Fichier");
Menu menuFichier = new Menu(shell, SWT.DROP_DOWN);
MenuItem optionOuvrir = new MenuItem(menuFichier, SWT.CASCADE);
optionOuvrir.setText("Ouvrir");
MenuItem optionFermer = new MenuItem(menuFichier, SWT.CASCADE);
optionFermer.setText("Fermer");
MenuItem optionCheck = new MenuItem(menuFichier, SWT.CHECK);
optionCheck.setText("Check");
optionFichier.setMenu(menuFichier);
MenuItem optionAide = new MenuItem(menu, SWT.CASCADE);
optionAide.setText("Aide");
shell.setMenuBar(menu);
```

SEPARATOR : une option de menu sous la forme d'un séparateur

```
Menu menu = new Menu(shell, SWT.BAR);
MenuItem optionFichier = new MenuItem(menu, SWT.CASCADE);
optionFichier.setText("Fichier");
Menu menuFichier = new Menu(shell, SWT.DROP_DOWN);
MenuItem optionOuvrir = new MenuItem(menuFichier, SWT.PUSH);
optionOuvrir.setText("Ouvrir");
MenuItem optionSeparator = new MenuItem(menuFichier, SWT.SEPARATOR);
MenuItem optionFermer = new MenuItem(menuFichier, SWT.PUSH);
optionFermer.setText("Fermer");
optionFichier.setMenu(menuFichier);
MenuItem optionAide = new MenuItem(menu, SWT.CASCADE);
optionAide.setText("Aide");
```

```
shell.setMenuBar(menu);
```

La méthode setText() permet de préciser le libellé de l'option de menu.

La méthode setAccelerator() permet de préciser un raccourci clavier.

```
Menu menuFichier = new Menu(shell, SWT.DROP_DOWN);
MenuItem optionOuvrir = new MenuItem(menuFichier, SWT.PUSH);
optionOuvrir.setText("&Ouvrir\tCtrl+O");
optionOuvrir.setAccelerator(SWT.CTRL+'O');
```

15.6.5. Les contrôles de sélection ou d'affichage d'une valeur

SWT propose un contrôle pour l'affichage d'une barre de progression et deux contrôles pour la sélection d'une valeur numérique dans une plage de valeur.

15.6.5.1. La classe ProgressBar

Ce contrôle est une barre de progression.

La classe ProgressBar possède plusieurs styles : BORDER, INDETERMINATE, SMOOTH, HORIZONTAL, VERTICAL

HORIZONTAL :

```
ProgressBar progressbar = new ProgressBar(shell,
  SWT.HORIZONTAL);
progressbar.setMinimum(1);
progressbar.setMaximum(100);
progressbar.setSelection(40);
progressbar.setSize(200,20);
```

SMOOTH :

```
ProgressBar progressbar = new ProgressBar(shell,
  SWT.HORIZONTAL | SWT.SMOOTH);
```

INDETERMINATE : la barre de progression s'incrémente automatiquement et revient au debut indéfiniment

```
ProgressBar progressbar = new ProgressBar(shell,
  SWT.INDETERMINATE);
```

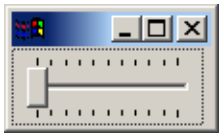
Les méthodes setMinimum() et setMaximum() permettent respectivement de préciser la valeur minimale et la valeur maximale du contrôle.

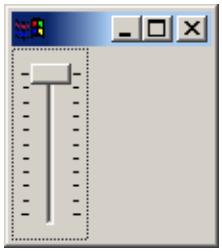
La méthode setSelection() permet de positionner la valeur courante de l'indicateur.

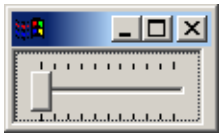
15.6.5.2. La classe Scale

Ce contrôle permet de sélectionner une valeur dans une plage valeur numérique.

La classe Scale possède trois styles : BORDER, HORIZONTAL, VERTICAL

HORZONTAL :
 	
`Scale scale = new Scale(shell,SWT.HORIZONTAL);`
`scale.setSize(100,40);`
	

VERTICAL :
 	
`Scale scale = new Scale(shell,SWT.VERTICAL);`
`scale.setSize(40,100);`
	

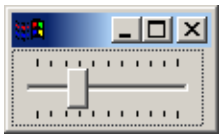
BORDER :
 	
`Scale scale = new Scale(shell,SWT.BORDER);`
`scale.setSize(100,40);`
	

Les méthodes setMinimum() et setMaximum() permettent respectivement de préciser la valeur minimale et la valeur maximale du contrôle.

La méthode setSelection() permet de positionner le curseur dans la plage de valeur à la valeur fournie en paramètre.

La méthode setPageIncrement() permet de préciser la valeur fourni en paramètre d'incrémentation d'une page

Exemple :

Scale scale = new Scale(shell,SWT.HORIZONTAL);
 	
`scale.setSize(100,40);`
`scale.setMinimum(1);`
`scale.setMaximum(100);`
`scale.setSelection(30);`
`scale.setPageIncrement(10);`
	

15.6.5.3. La classe Slider

Ce contrôle permet de sélectionner une valeur dans une plage valeur numérique.

La classe Slider possède trois styles : BORDER, HORIZONTAL, VERTICAL

BORDER :
 	
`Slider slider = new Slider(shell, SWT.BORDER);`
`slider.setSize(200,20);`
	

VERTICAL :
 	
`Slider slider = new Slider(shell, SWT.VERTICAL);`
`slider.setSize(20,200);`
	

Les méthodes setMinimum() et setMaximum() permettent respectivement de préciser la valeur minimale et la valeur maximale du contrôle.

La méthode setSelection() permet de positionner le curseur dans la plage de valeur à la valeur fournie en paramètre.

La méthode setPageIncrement() permet de préciser la valeur fourni en paramètre d'incrémentation d'une page

La méthode setThumb() permet de préciser la taille du curseur.

Exemple :

Slider slider = new Slider(shell,SWT.HORIZONTAL);

```
slider.setMinimum(1);
slider.setMaximum(110);
slider.setSelection(30);
slider.setThumb(10);
slider.setSize(100,20);
```

15.6.6. Les contrôles de type « onglets »

SWT propose la création de composants de type onglets mettant en oeuvre deux classes : TabFolder et TabItem

15.6.6.1. La classe TabFolder

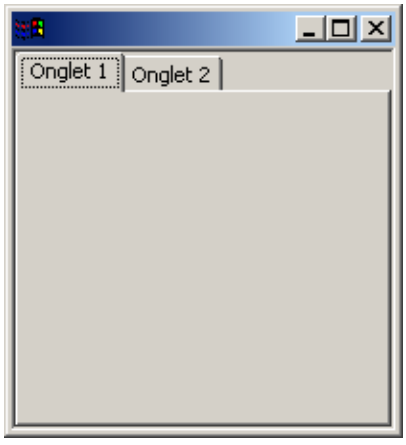
Ce contrôle est un ensemble d'onglets.

NONE :

```
TabFolder tabfolder = new TabFolder(shell, SWT.NONE);
tabfolder.setSize(200,200);
TabItem onglet1 = new TabItem(tabfolder, SWT.NONE);
onglet1.setText("Onglet 1");
TabItem onglet2 = new TabItem(tabfolder, SWT.NONE);
onglet2.setText("Onglet 2");
```

BORDER : un ensemble d'onglet avec une bordure

```
TabFolder tabfolder = new TabFolder(shell, SWT.BORDER);
```



15.6.6.2. La classe TabItem

Ce contrôle est un onglet d'un ensemble d'onglets

Il n'est possible d'insérer qu'un seul contrôle dans un onglet grâce la commande setControl(). Rl est ainsi pratique de regrouper les différents dans un contrôle de type Composite.

Exemple :

```
TabFolder tabfolder = new TabFolder(shell, SWT.NONE);
tabfolder.setSize(200,100);
TabItem onglet1 = new TabItem(tabfolder, SWT.NONE);
onglet1.setText("Onglet 1");
TabItem onglet2 = new TabItem(tabfolder, SWT.NONE);
onglet2.setText("Onglet 2");

Composite pageOnglet1 = new Composite(tabfolder, SWT.NONE);
Text text1 = new Text(pageOnglet1, SWT.BORDER);
text1.setText("mon texte");
text1.setBounds(10,10,100,25);
Button button1 = new Button(pageOnglet1, SWT.BORDER);
button1.setText("Valider");
button1.setBounds(10,40,100,25);
onglet1.setControl(pageOnglet1);
```

15.6.7. Les contrôles de type « tableau »

SWT permet la création d'un contrôle de type tableau pour afficher et sélectionner des données en mettant en oeuvre trois classes : Table, TableColumn et TableItem.

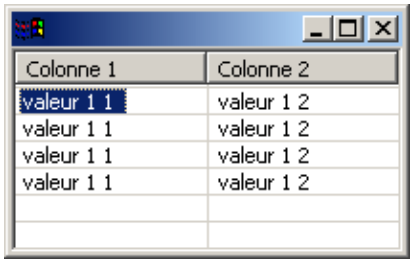
15.6.7.1. La classe Table

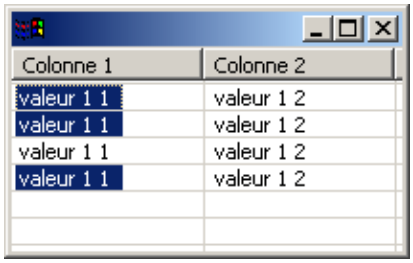
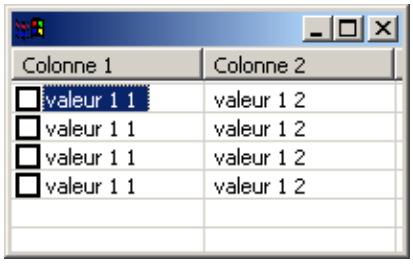
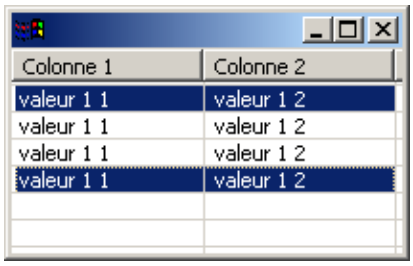
Ce contrôle permet d'afficher et de sélectionner des éléments sous la forme d'un tableau.

La classe Table possède plusieurs styles : BORDER, H_SCROLL, V_SCROLL, SINGLE, MULTI, CHECK, FULL_SELECTION, HIDE_SELECTION

BORDER :

```
Table table = new Table(shell, SWT.BORDER);
table.setSize(204,106);
TableColumn colonne1 = new TableColumn(table, SWT.LEFT);
colonne1.setText("Colonne 1");
colonne1.setWidth(100);
TableColumn colonne2 = new TableColumn(table, SWT.LEFT);
colonne2.setText("Colonne 2");
colonne2.setWidth(100);
table.setHeaderVisible(true);
table.setLinesVisible(true);
TableItem ligne1 = new TableItem(table,SWT.NONE);
ligne1.setText(new String[] {"valeur 1 1","valeur 1 2"});
TableItem ligne2 = new TableItem(table,SWT.NONE);
ligne2.setText(new String[] {"valeur 1 1","valeur 1 2"});
TableItem ligne3 = new TableItem(table,SWT.NONE);
ligne3.setText(new String[] {"valeur 1 1","valeur 1 2"});
TableItem ligne4 = new TableItem(table,SWT.NONE);
ligne4.setText(new String[] {"valeur 1 1","valeur 1 2"});
```



MULTI : permet la selection de plusieurs élements dans la table `Table table = new Table(shell, SWT.BORDER);`	
CHECK : une table avec une case à cocher pour chaque ligne `Table table = new Table(shell, SWT.CHECK);`	
FULL_SELECTION : la ou les lignes sélectionnées sont entièrement mises en valeur `Table table = new Table(shell, SWT.MULTI | SWT.FULL_SELECTION);`	
HIDE_SELECTION : seule la première colonne sélectionnées sont mises en valeur	

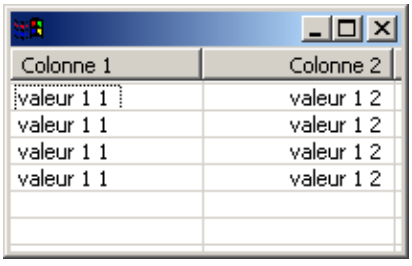
La méthode setHeaderVisible() permet de préciser si l'en tête de la table doit être affichée ou non : par défaut sa valeur est non affichée (false).

La méthode setLinesVisible() permet de préciser si lignes de la table doivent être affichées ou non : par défaut sa valeur est non affiché (false).

15.6.7.2. La classe TableColumn

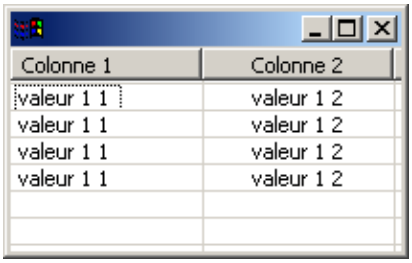
Ce contrôle est une colonne d'un contrôle Table

La classe TableColumn possède trois styles : LEFT, RIGHT, CENTER

LEFT : alignement de la colonne sur la gauche (valeur par défaut)	
RIGHT : alignement de la colonne sur la droite `TableColumn colonne2 = new TableColumn(table, SWT.RIGHT);`	

CENTER : alignement centré de la colonne

```
TableColumn colonne2 = new TableColumn(table, SWT.CENTER);
```



Bizaremment seul le style LEFT semble pouvoir s'appliquer à la première colonne de la table.

La méthode setWidth() permet de préciser la largeur de la colonne

La méthode setText() permet de préciser le libellé d'en tête de la colonne

La méthode setResizable() permet de préciser si la colonne peut être redimmensionnée ou non.

15.6.7.3. La classe TableItem

Ce contrôle est une ligne d'un contrôle Table

La classe TableItem ne possède aucun style.

Il existe plusieurs surcharges de la méthode setText() pour fournir à chaque ligne les données de ces colonnes.

Une surcharge de cette méthode permet de fournir les données sous la forme d'un tableau de chaînes de caractères.

Exemple :

```
ligne1.setText(new String[] {"valeur 1 1","valeur 1 2"});
```

Une autre surcharge de cette méthode permet de préciser le numéro de la colonne et le texte. La première colonne possède le numéro 0.

Exemple : modifier la valeur de la première cellule de la ligne

```
ligne4.setText(0 ,"valeur 2 2");
```

La méthode setCheck() permet de cocher ou non la case associée à la ligne si la table possède le style CHECK.

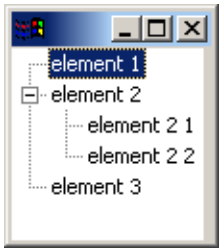
15.6.8. Les contrôles de type « arbre »

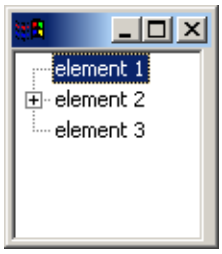
SWT permet la création d'un composant de type arbre en mettant en oeuvre les classes Tree et TreeItem.

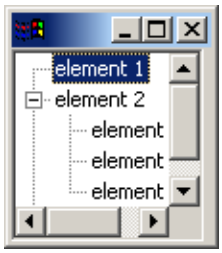
15.6.8.1. La classe Tree

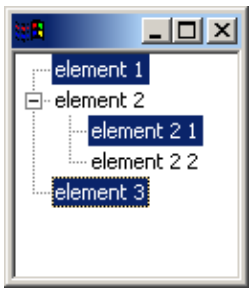
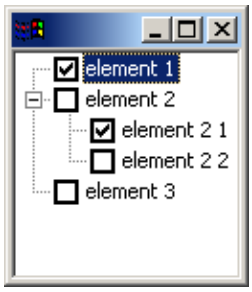
Ce contrôle affiche et permet de sélectionner des données sous la forme d'une arborescence

La classe Tree possède plusieurs styles : BORDER, H_SCROLL, V_SCROLL, SINGLE, MULTI, CHECK

SINGLE : un arbre avec sélection unique
 	
`Tree tree = new Tree(shell, SWT.SINGLE);`
`TreeItem tree_1 = new TreeItem(tree, SWT.NONE);`
`tree_1.setText("element 1");`
`TreeItem tree_2 = new TreeItem(tree, SWT.NONE);`
`tree_2.setText("element 2");`
`TreeItem tree_2_1 = new TreeItem(tree_2, SWT.NONE);`
`tree_2_1.setText("element 2 1");`
`TreeItem tree_2_2 = new TreeItem(tree_2, SWT.NONE);`
`tree_2_2.setText("element 2 2");`
`TreeItem tree_3 = new TreeItem(tree, SWT.NONE);`
`tree_3.setText("element 3");`
`tree.setSize(100, 100);`
	

BORDER : arbre avec une bordure `Tree tree = new Tree(shell, SWT.SINGLE | SWT.BORDER);`	

H_SCROLL et V_SCROLL : arbre avec si nécessaire une barre de défilement respectivement horizontal et vertical
 	
`Tree tree = new Tree(shell, SWT.SINGLE | SWT.BORDER |`
`  SWT.H_SCROLL | SWT.V_SCROLL);`
	

MULTI : un arbre avec sélection multiple possible `Tree tree = new Tree(shell, SWT.MULTI | SWT.BORDER);`	
CHECK : un arbre avec une case à cocher devant chaque élément `Tree tree = new Tree(shell, SWT.CHECK | SWT.BORDER);`	

15.6.8.2. La classe TreeItem

Ce contrôle est un élément d'une arborescence

Cette classe ne possède pas de style particulier.

Pour ajouter un élément racine à l'arbre, il suffit de passer l'arbre en tant qu'élément conteneur dans le constructeur.

Pour ajouter un élément fils à un élément, il suffit de passer l'élément père en tant qu'élément conteneur dans le constructeur.

Il existe un constructeur qui attend un troisième paramètre permettant de préciser la position de l'élément.

Exemple :

```
Tree tree = new Tree(shell, SWT.SINGLE);
TreeItem tree_1 = new TreeItem(tree, SWT.NONE);
tree_1.setText("element 1");
TreeItem tree_2 = new TreeItem(tree, SWT.NONE);
tree_2.setText("element 2");
TreeItem tree_2_1 = new TreeItem(tree_2, SWT.NONE);
tree_2_1.setText("element 2 1");
TreeItem tree_2_2 = new TreeItem(tree_2, SWT.NONE);
tree_2_2.setText("element 2 2");
TreeItem tree_3 = new TreeItem(tree, SWT.NONE);
tree_3.setText("element 3");
tree.setSize(100, 100);
```

15.6.9. La classe ScrollBar

Ce contrôle est une barre de défilement

La classe ScrollBar possède deux styles : HORIZONTAL, VERTICAL

15.6.10. Les contrôles pour le graphisme

SWT permet de dessiner des formes graphiques en mettant en oeuvre la classe GC et la classe Canvas.

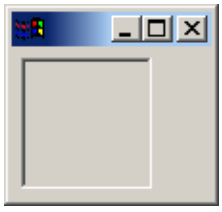
15.6.10.1. La classe Canvas

Ce contrôle est utilisé pour dessiner des formes graphiques

La classe Canvas définit plusieurs styles : BORDER, H_SCROLL, V_SCROLL, NO_BACKGROUND, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE, NO_RADIO_GROUP

BORDER : une zone de dessin avec bordure

```
Canvas canvas = new Canvas(shell, SWT.BORDER);
canvas.setSize(200,200);
```



15.6.10.2. La classe GC

Cette classe encapsule un contexte graphique dans lequel il va être possible de dessiner des formes graphiques.

Pour réaliser ces opérations, la classe GC propose de nombreuses méthodes.

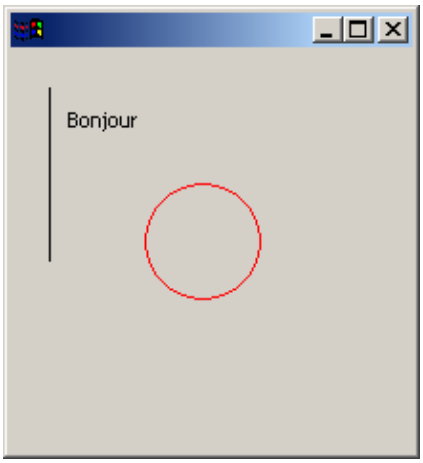
Attention : il est important d'appeler la méthode open() de la fenêtre avant de réaliser des opérations de dessin sur le contexte.

Ne pas oublier le libérer les ressources allouées à la classe GC en utilisant la méthode dispose() si l'objet de GC est explicitement instancié dans le code.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
```

```
import org.eclipse.swt.graphics.*;

public class TestSWT21 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setSize(420,420);

    Canvas canvas = new Canvas(shell, SWT.NONE);
    canvas.setSize(200,200);
    canvas.setLocation(10,10);
    shell.pack();
    shell.open();

    GC gc = new GC(canvas);
    gc.drawText("Bonjour",20,20);
    gc.drawLine(10,10,10,100);
    gc.setForeground(display.getSystemColor(SWT.COLOR_RED));
    gc.drawOval(60,60,60,60);
    gc.dispose();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }
}
```



Dans cette exemple, le dessin est réalisé une seule fois au démarrage, il n'est donc pas redessiné si nécessaire (fenêtre partiellement ou complètement masquée, redimensionnement, ...). Pour résoudre ce problème, il faut mettre les opérations de dessin en réponse à un événement de type PaintListener.

15.6.10.3. La classe Color

Cette classe encapsule une couleur définie dans le système graphique.

Elle possède deux constructeurs qui attendent en paramètre l'objet de type Display et soit un objet de type RGB, soit trois entiers représentant les valeurs des couleurs rouge, vert et bleu.

La classe RGB encapsule simplement les trois entiers représentant les valeurs des couleurs rouge, vert et bleu.

Exemple :

```
Color couleur = new Color(display,155,0,0);
Color couleur = new Color(display, new RGB(155,0,0));
```

Remarque : il ne faut pas oublier d'utiliser la méthode dispose() pour libérer les ressources du système allouées à cet objet une fois que celui ci n'est plus utilisé.

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;

public class TestSWT22 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    shell.setSize(200, 200);
    Color couleur = new Color(display,155,0,0);
    shell.setBackground(couleur);
    shell.open();

   while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    couleur.dispose();
    display.dispose();
  }
}
```

15.6.10.4. La classe Font

Cette classe encapsule une police de caractère définie dans le système graphique.

La classe Font peut utiliser plusieurs styles : NORMAL, BOLD et ITALIC

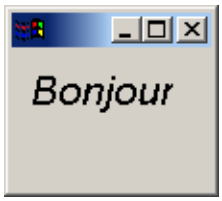
Il existe plusieurs constructeurs dont le plus simple à utiliser nécessite en paramètre l'objet display, le nom de la police (celle ci doit être présente sur le système), la taille et le style.

Remarque : il ne faut pas oublier d'utiliser la méthode dispose() pour libérer les ressources du système allouées à cet objet une fois que celui ci n'est plus utilisé.

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.*;

public class TestSWT23 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    Font font = new Font(display, "Arial", 16, SWT.ITALIC);
    Label label = new Label(shell, SWT.NONE);
    label.setFont(font);
    label.setText("Bonjour");
    label.setLocation(10, 10);
    label.pack();
    shell.setSize(100, 100);
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
```

```
    font.dispose();
    display.dispose();
  }
}
```



15.6.10.5. La classe Image

Cette classe encapsule une image au format BMP, ICO, GIF, JPEG ou PNG.

La classe Image possède plusieurs constructeurs dont le plus simple à utiliser est celui nécessitant en paramètres l'objet Display et une chaîne de caractères contenant le chemin vers le fichier de l'image

Remarque : il ne faut pas oublier d'utiliser la méthode dispose() pour libérer les ressources du système allouées à cet objet une fois que celui ci n'est plus utilisé.

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.*;

public class TestSWT24 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    Image image = new Image(display, "btn1.bmp");
    Button bouton = new Button(shell, SWT.FLAT);
    bouton.setImage(image);
    bouton.setBounds(10, 10, 50, 50);
    shell.setSize(100, 100);
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    image.dispose();
    display.dispose();
  }
}
```

Si l'application doit être packagée dans un fichier jar, incluant les images utiles, il faut utiliser la getResourceAsStream() du classloader pour charger l'image.

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.*;
import java.io.*;

public class TestSWT25 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    InputStream is = TestSWT25.class.getResourceAsStream("btn1.bmp");
    Image image = new Image(display, is);
    Button bouton = new Button(shell, SWT.FLAT);
    bouton.setImage(image);
    bouton.setBounds(10, 10, 50, 50);
    shell.setSize(100, 100);
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    image.dispose();
    display.dispose();
  }
}
```

15.7. 1.7 Les conteneurs

Ce type de composant permettent de contenir d'autres contrôles.

15.7.1. Les conteneurs de base

SWT propose deux contrôles de ce type : Composite et Group.

15.7.1.1. La classe Composite

Ce contrôle est un conteneur pour d'autres contrôles.

Ce contrôle possède les styles particuliers suivants : BORDER, H_SCROLL et V_SCROLL

BORDER : permet la présence d'une bordure autour du composant `Composite composite = new Composite(shell, SWT.NONE);`	
H_SCROLL : permet la présence d'une barre de défilement horizontal `Composite composite = new Composite(shell, SWT.H_SCROLL);`	
V_SCROLL : permet la présence d'une barre de defilement vertical `Composite composite = new Composite(shell, SWT.V_SCROLL);`	

Les contrôles sont ajoutés au contrôle Composite de la même façon que dans un objet de type Shell en précisant

simplement que le conteneur est l'objet de type Composite.

La position indiquée pour les contrôles inclus dans le Composite est relative par rapport à l'objet Composite.

Exemple complet :

```
import org.eclipse.swt.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.*;

public class TestSWT2 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    Composite composite = new Composite(shell, SWT.BORDER);
    Color couleur = new Color(display,131,133,131);
    composite.setBackground(couleur);
    Label label = new Label(composite, SWT.NONE);
    label.setBackground(couleur);
    label.setText("Saisir la valeur");
    label.setBounds(10, 10, 100, 25);
    Text text = new Text(composite, SWT.BORDER);
    text.setText("mon texte");
    text.setBounds(10, 30, 100, 25);
    Button button = new Button(composite, SWT.BORDER);
    button.setText("Valider");
    button.setBounds(10, 60, 100, 25);
    composite.setSize(140,140);

    shell.pack();
    shell.open();
    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    couleur.dispose();
    display.dispose();
  }
}
```

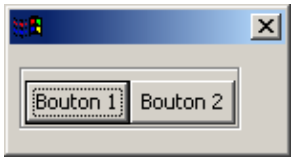
15.7.1.2. La classe Group

Ce contrôle permet de regrouper d'autres contrôles en les entourant d'une bordure et éventuellement d'un libellé.

La classe Group possède plusieurs styles : BORDER, SHADOW_ETCHED_IN, SHADOW_ETCHED_OUT, SHADOW_IN, SHADOW_OUT, SHADOW_NONE

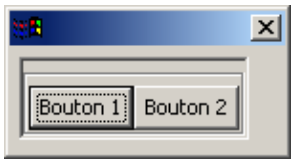
NONE : un cadre simple

```
Group group = new Group(shell, SWT.NONE);
group.setLayout (new FillLayout ());
button button1 = new Button(group, SWT.NONE);
button1.setText("Bouton 1");
Button button2 = new Button(group, SWT.NONE);
button2.setText("Bouton 2");
```



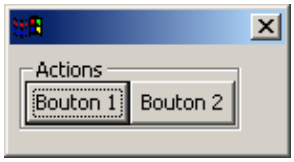
BORDER : un cadre simple avec une bordure

```
Group group = new Group(shell, SWT.BORDER);
```



La méthode setText() permet de préciser un titre affiché en haut à gauche du cadre.

```
Group group = new Group(shell, SWT.NONE);
group.setLayout (new FillLayout ());
group.setText ("Actions");
Button button1 = new Button(group, SWT.NONE);
button1.setText("Bouton 1");
Button button2 = new Button(group, SWT.NONE);
button2.setText("Bouton 2");
```



15.7.2. Les contrôles de type « barre d'outils »

SWT permet de créer des barres d'outils fixes et barres d'outils flottantes.

15.7.2.1. La classe ToolBar

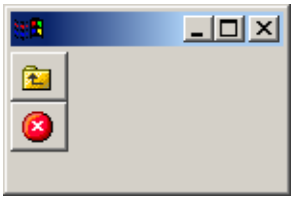
Ce contrôle est une barre d'outils

La classe ToolBar possède plusieurs styles : BORDER, FLAT, WRAP, RIGHT, SHADOW_OUT HORIZONTAL, VERTICAL

HORIZONTAL : une barre d'outils horizontale (style par défaut)

```
shell.setSize(150, 100);
ToolBar toolbar = new ToolBar(shell, SWT.HORIZONTAL);
toolbar.setSize(shell.getSize().x, 35);
toolbar.setLocation(0, 0);

Image imageBtn1 = new Image(display, "btn1.bmp");
ToolItem btn1 = new ToolItem(toolbar, SWT.PUSH);
btn1.setImage(imageBtn1);

Image imageBtn2 = new Image(display, "btn2.bmp");
ToolItem btn2 = new ToolItem(toolbar, SWT.PUSH);
btn2.setImage(imageBtn2);

shell.open();
while (!shell.isDisposed())
  if (!display.readAndDispatch())
    display.sleep();

imageBtn1.dispose();
imageBtn2.dispose();
display.dispose();
```

FLAT : une barre d'outils sans effet 3D `ToolBar toolbar = new ToolBar(shell, SWT.FLAT);`	

VERTICAL : une barre d'outils vertical
 	
`ToolBar toolbar = new ToolBar(shell, SWT.VERTICAL);`
`toolbar.setSize(35, shell.getSize().y);`
	

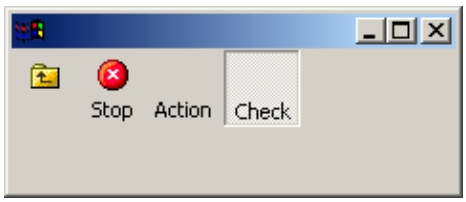
BORDER : une barre d'outils avec une bordure `ToolBar toolbar = new ToolBar(shell, SWT.BORDER);`	

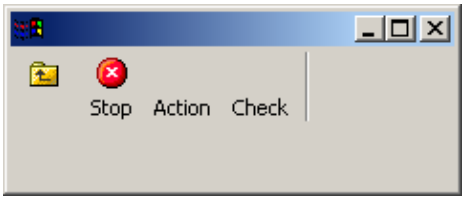
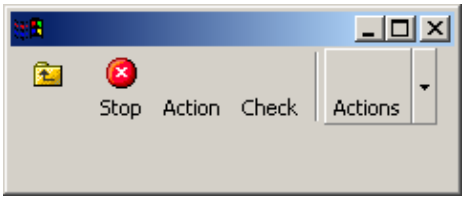
15.7.2.2. La classe ToolItem

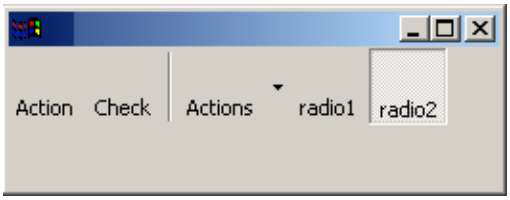
Ce contrôle est une élément d'une barre d'outils

La classe ToolItem possède styles : PUSH, CHECK, RADIO, SEPARATOR, DROP_DOWN

PUSH : un bouton simple
 	
`shell.setSize(240, 100);`
`ToolBar toolbar = new ToolBar(shell, SWT.FLAT);`
`toolbar.setSize(shell.getSize().x, 40);`
`toolbar.setLocation(0, 0);`
 	
`Image imageBtn1 = new Image(display, "btn1.bmp");`
`ToolItem btn1 = new ToolItem(toolbar, SWT.PUSH);`
`btn1.setImage(imageBtn1);`
 	
`Image imageBtn2 = new Image(display, "btn2.bmp");`
`ToolItem btn2 = new ToolItem(toolbar, SWT.PUSH);`
`btn2.setImage(imageBtn2);`
`btn2.setText("Stop");`
 	
`ToolItem btn3= new ToolItem(toolbar, SWT.PUSH);`
`btn3.setText("Action");`
`shell.open();`
 	
`while (!shell.isDisposed())`
`  if (!display.readAndDispatch())`
`    display.sleep();`
 	
`imageBtn1.dispose();`
`imageBtn2.dispose();`
`display.dispose();`
	

CHECK : un bouton qui peut conserver son état enfoncé
 	
`ToolItem btn4= new ToolItem(toolbar, SWT.CHECK);`
`btn4.setText("Check");`
	

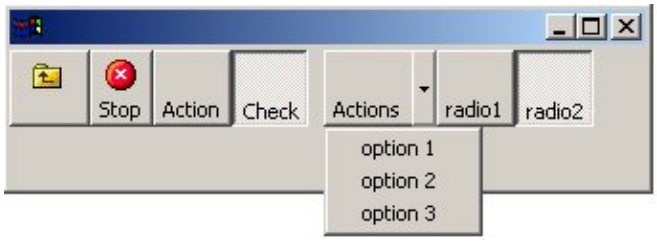
SEPARATOR : un séparateur `ToolItem btn5= new ToolItem(toolbar, SWT.SEPARATOR);`	
DROP_DOWN : un bouton avec une petite flèche vers le bas	

RADIO : un bouton dont un seul d'un même ensemble peut être sélectionné (un ensemble est défini par des boutons de type radio qui sont adjacents)
 	
`ToolItem btn7= new ToolItem(toolbar, SWT.RADIO);`
`btn7.setText("radio1");`
`ToolItem btn8= new ToolItem(toolbar, SWT.RADIO);`
`btn8.setText("radio2");`
	

Exemple :

```
shell.setSize(340, 100);
final ToolBar toolbar = new ToolBar(shell, SWT.HORIZONTAL);
toolbar.setSize(shell.getSize().x, 45);
toolbar.setLocation(0, 0);

Image imageBtn1 = new Image(display, "btn1.bmp");
ToolItem btn1 = new ToolItem(toolbar, SWT.PUSH);
btn1.setImage(imageBtn1);

Image imageBtn2 = new Image(display, "btn2.bmp");
ToolItem btn2 = new ToolItem(toolbar, SWT.PUSH);
btn2.setImage(imageBtn2);
btn2.setText("Stop");

ToolItem btn3 = new ToolItem(toolbar, SWT.PUSH);
btn3.setText("Action");

ToolItem btn4 = new ToolItem(toolbar, SWT.CHECK);
btn4.setText("Check");

ToolItem btn5 = new ToolItem(toolbar, SWT.SEPARATOR);

final ToolItem btn6 = new ToolItem(toolbar, SWT.DROP_DOWN);
btn6.setText("Actions");

final Menu menu = new Menu(shell, SWT.POP_UP);
MenuItem menu1 = new MenuItem(menu, SWT.PUSH);
menu1.setText("option 1");
MenuItem menu2 = new MenuItem(menu, SWT.PUSH);
menu2.setText("option 2");
MenuItem menu3 = new MenuItem(menu, SWT.PUSH);
menu3.setText("option 3");

btn6.addListener(SWT.Selection, new Listener() {
  public void handleEvent(Event event) {
    if (event.detail == SWT.ARROW) {
      Rectangle rect = btn6.getBounds();
      Point pt = new Point(rect.x, rect.y + rect.height);
      pt = toolbar.toDisplay(pt);
      menu.setLocation(pt.x, pt.y);
      menu.setVisible(true);
    }
  }
});
```

```
ToolItem btn7 = new ToolItem(toolbar, SWT.RADIO);
btn7.setText("radio1");

ToolItem btn8 = new ToolItem(toolbar, SWT.RADIO);
btn8.setText("radio2");

shell.open();
while (!shell.isDisposed())
  if (!display.readAndDispatch())
    display.sleep();

imageBtn1.dispose();
imageBtn2.dispose();
display.dispose();
```



15.7.2.3. Les classes CoolBar et CoolItem

La classe CoolBar est un élément d'une barre d'outils repositionnable. Ce contrôle doit être utilisé avec un ou plusieurs contrôles CoolItem qui représentent un élément de la barre.

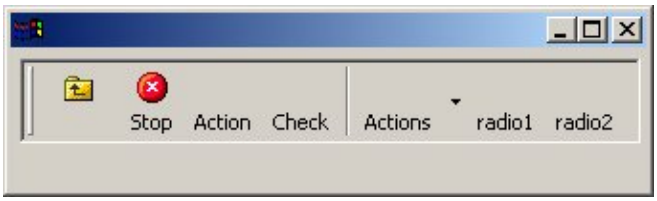
La classe CoolItem est un élément d'une barre de type CoolBar

Le plus simple est d'associer une barre d'outils de type ToolBar à un de ces éléments en utilisant la méthode setControl() de la classe CoolItem.

Exemple : utilisation de la barre d'outils définie dans la section précédente :

```
shell.setLayout(new GridLayout());

shell.setSize(340, 100);
CoolBar coolbar = new CoolBar(shell, SWT.BORDER);
final ToolBar toolbar = new ToolBar(coolbar, SWT.FLAT);

Image imageBtn1 = new Image(display, "btn1.bmp");
ToolItem btn1 = new ToolItem(toolbar, SWT.PUSH);
btn1.setImage(imageBtn1);

Image imageBtn2 = new Image(display, "btn2.bmp");
ToolItem btn2 = new ToolItem(toolbar, SWT.PUSH);
btn2.setImage(imageBtn2);
btn2.setText("Stop");

ToolItem btn3 = new ToolItem(toolbar,SWT.PUSH);
btn3.setText("Action");

ToolItem btn4 = new ToolItem(toolbar,SWT.CHECK);
btn4.setText("Check");

ToolItem btn5 = new ToolItem(toolbar, SWT.SEPARATOR);

final ToolItem btn6 = new ToolItem(toolbar, SWT.DROP_DOWN);
btn6.setText("Actions");

final Menu menu = new Menu(shell, SWT.POP_UP);
MenuItem menu1 = new MenuItem(menu, SWT.PUSH);
menu1.setText("option 1");
MenuItem menu2 = new MenuItem(menu, SWT.PUSH);
```

```
menu2.setText("option2");
MenuItem menu3 = new MenuItem(menu, SWT.PUSH);
menu3.setText("option3");

btn6.addListener(SWT.Selection, new Listener() {
  public void handleEvent(Event event) {
    if (event.detail == SWT.ARROW) {
      Rectangle rect = btn6.getBounds();
      Point pt = new Point(rect.x, rect.y + rect.height);
      pt = toolbar.toDisplay(pt);
      menu.setLocation(pt.x, pt.y);
      menu.setVisible(true);
    }

  }
});

ToolItem btn7 = new ToolItem(toolbar, SWT.RADIO);
btn7.setText("radio1");

ToolItem btn8 = new ToolItem(toolbar, SWT.RADIO);
btn8.setText("radio2");

CoolItem coolItem = new CoolItem(coolbar, SWT.NONE);
coolItem.setControl(toolbar);
Point size = toolbar.computeSize(SWT.DEFAULT, SWT.DEFAULT);
coolItem.setPreferredSize(coolItem.computeSize(size.x, size.y));

shell.open();

while (!shell.isDisposed())
  if (!display.readAndDispatch())
    display.sleep();

imageBtn1.dispose();
imageBtn2.dispose();
```



La classe CoolItem possède une méthode setLocked() qui attend un booléen en paramètre précisant si le contrôle peut être déplacé ou non. Cette méthode doit être appelée lors d'un clic sur un bouton de la barre pour empécher le déplacement de celle ci lors du clic sur le bouton.

15.8. La gestion des erreurs

Lors de l'utilisation de l'API SWT, des exceptions de trois types peuvent être levées :

- IllegalArgumentException : un argument fourni à une méthode est invalide
- SWTException : cette exception est levée lors d'une erreur non fatale. Le code de l'erreur et le message de l'exception permettent d'obtenir des précisions sur l'exception
- SWTError : cette exception est levée lors d'une erreur fatale

15.9. Le positionnement des contrôles

15.9.1. 1.9.1 Le positionnement absolu

Dans ce mode, il faut préciser pour chaque composant, sa position et sa taille. L'inconvénient de ce mode de positionnement est qu'il réagit très mal à un changement de la taille du conteneur des composants.

15.9.2. 1.9.2 La positionnement relatif avec les LayoutManager

SWT propose un certain nombre de gestionnaires de positionnement de contrôles (layout manager). Ceux ci sont regroupés dans le package org.eclipse.swt.layout.

Le grand avantage de ce mode de positionnement est de laisser au LayoutManager utilisé le soin de positionner et de dimensionner chaque composant en fonction de ces règles et des paramètres qui lui sont fournis.

SWT définit quatre gestionnaires de positionnement :

- RowLayout pour un arrangement simple de modèle de mot−sur−un−page des composants
- FillLayout pour les composants égal−classés qui occupent une colonne ou une rangée simple
- GridLayout pour une grille rectangulaire des cellules composantes
- FormLayout pour un positionnement plus précis mais aussi plus compliqué des composants.

Pour personnaliser finement l'arrangement des composants, des informations complémentaires peuvent être associées à chacun d'eux en utilisant un objet dédié du type RowData, GridData ou FormData respectivement pour les gestionnaires de positionnement RowLayout, GridLayout et FormLayout.

15.9.2.1. 1.9.2.1 FillLayout

Le FillLayout est le gestionnaire de positionnement le plus simple : il organise les composants dans une colonne ou une rangée. L'espace entre les composants est calculé automatiquement par la classe FillLayout.

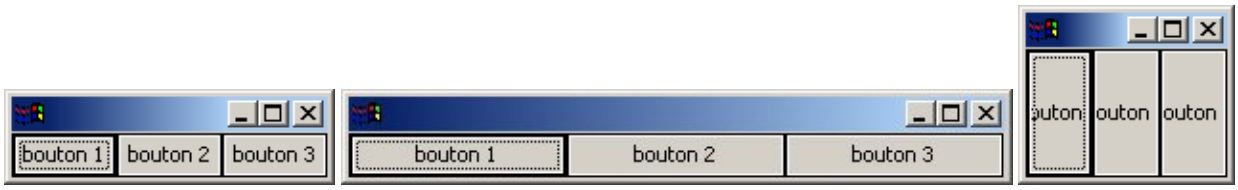
La classe FillLayout peut utiliser deux styles : SWT.HORIZONTAL (par défaut) et SWT.VERTICAL pour préciser le mode d'alignement

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.*;

public class TestSWT27 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    shell.setLayout(new RowLayout());
    Button bouton1 = new Button(shell, SWT.FLAT);
    bouton1.setText("bouton 1");
    Button bouton2 = new Button(shell, SWT.FLAT);
    bouton2.setText("bouton 2");
    Button bouton3 = new Button(shell, SWT.FLAT);
    bouton3.setText("bouton 3");

    shell.pack();
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    display.dispose();
```

```
  }
}
```

Voici différents aperçus en cas de modification de la taille de la fenêtre.



Il n'est pas possible de mettre un espace entre le bord du conteneur et les composants avec ce gestionnaire. Il n'est pas non plus possible pour ce gestionnaire de mettre des composants sur plusieurs colonnes ou rangées.

15.9.2.2. 1.9.2.2 RowLayout

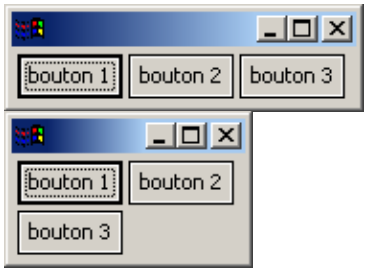
Ce gestionnaire propose d'arranger les composants en rangée horizontale ou verticale. Il possède des paramètres permettant de préciser une marge, un espace, une rupture et une compression.

Propriété	Valeur par défaut	Rôle
wrap	true	demande de faire une rupture dans la rangée si il n'y a plus de place false :  true : 
pack	true	demande à chaque composants de prendre sa taille préférée
justify	false	justification des composants 
type	SWT.HORIZONTAL	type de mise en forme SWT.HORIZONTAL ou SWT.VERTICAL SWT.VERTICAL : 
marginLeft	3	taille en pixel de la marge gauche
marginTop	3	taille en pixel de la marge haute
marginRight	3	taille en pixel de la marge droite
marginBottom	3	taille en pixel de la marge basse
spacing	3	taille en pixel entre deux cellules

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.*;

public class TestSWT27 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    RowLayout rowlayout = new RowLayout();
    shell.setLayout(rowlayout);
    Button bouton1 = new Button(shell, SWT.FLAT);
    bouton1.setText("bouton 1");

    Button bouton2 = new Button(shell, SWT.FLAT);
    bouton2.setText("bouton 2");

    Button bouton3 = new Button(shell, SWT.FLAT);
    bouton3.setText("bouton 3");

    shell.pack();
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }
}
```

Voici différents aperçu en cas de modification de la taille de la fenêtre.



15.9.2.3. GridLayout

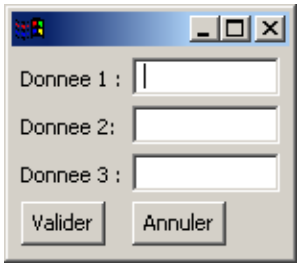
Ce gestionnaire permet d'arranger les composants dans une grille.

Ce gestionnaire possède plusieurs propriétés :

Propriété	Valeur par défaut	Rôle
horizontalSpacing	5	préciser l'espace horizontal entre chaque cellule
makeColumnsEqualWidth	false	donner à toute la colonne de la grille la même largeur
marginHeight	5	préciser la hauteur de la marge
marginWidth	5	préciser la largeur de la marge
numColumns	1	préciser le nombre de colonnes de la grille
verticalSpacing	5	préciser l'espace vertical entre chaque cellule

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.*;

public class TestSWT28 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    GridLayout gridLayout = new GridLayout();
    gridLayout.numColumns = 2;
    shell.setLayout(gridLayout);

    Label label1 = new Label(shell, SWT.NONE);
    label1.setText("Donnee 1 :");
    Text text1 = new Text(shell, SWT.BORDER);
    text1.setSize(200, 10);

    Label label2 = new Label(shell, SWT.NONE);
    label2.setText("Donnee 2:");
    Text text2 = new Text(shell, SWT.BORDER);
    text2.setSize(200, 10);

    Label label3 = new Label(shell, SWT.NONE);
    label3.setText("Donnee 3 :");
    Text text3 = new Text(shell, SWT.BORDER);
    text3.setSize(200, 10);

    Button button1 = new Button(shell, SWT.NONE);
    button1.setText("Valider");

    Button button2 = new Button(shell, SWT.NONE);
    button2.setText("Annuler");

    shell.pack();
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    display.dispose();
  }

}
```



Les paramètres liés à un composant d'une cellule particulière de la grille peuvent être précisés grâce à un objet de type GridData. Ces paramètres précisent le comportement du composant en cas de redimensionnement.

Il existe deux façons de créer un objet de type GridData :

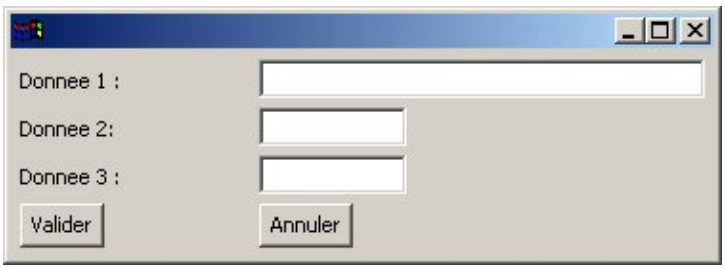
- instancier un objet de type GridData avec son constructeur sans paramètre et initialiser les proprités en utilisant les setters appropriés.
- instancier un objet de type GridData avec son constructeur attendant un style en paramètre

La méthode setLayoutData() permet de d'associer un objet GridData à un composant.

Attention : il ne faut pas utiliser plusieurs fois un objet de type GridData.

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.*;

public class TestSWT28 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);

    GridLayout gridLayout = new GridLayout();
    gridLayout.numColumns = 2;
    shell.setLayout(gridLayout);

    Label label1 = new Label(shell, SWT.NONE);
    label1.setText("Donnee 1 :");
    Text text1 = new Text(shell, SWT.BORDER);
    text1.setSize(200, 10);

    Label label2 = new Label(shell, SWT.NONE);
    label2.setText("Donnee 2:");
    Text text2 = new Text(shell, SWT.BORDER);
    text2.setSize(200, 10);

    Label label3 = new Label(shell, SWT.NONE);
    label3.setText("Donnee 3 :");
    Text text3 = new Text(shell, SWT.BORDER);
    text3.setSize(200, 10);

    Button button1 = new Button(shell, SWT.NONE);

    button1.setText("Valider");

    Button button2 = new Button(shell, SWT.NONE);
    button2.setText("Annuler");

    GridData data = new GridData();
    data.widthHint = 120;
    label1.setLayoutData(data);

    data = new GridData();
    data.widthHint = 220;
    text1.setLayoutData(data);

    shell.pack();
    shell.open();

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    display.dispose();
  }

}
```



15.9.2.4. FormLayout

Ce gestionnaire possède deux propriétés :

Propriété	Valeur par défaut	Rôle
marginHeight	0	préciser la hauteur de la marge
marginWidth	0	préciser la largeur de la marge

Ce gestionnaire impose d'associer à chaque composant un objet de type FormData qui va préciser les informations de positionnement et de comportement du composant.

15.10. La gestion des événements

La gestion de événements avec SWT est très similaire à celle proposée par l'API Swing car elle repose sur les Listeners. Ces Listeners doivent être ajoutés au contrôle en fonction des événements qu'ils doivent traiter.

Dès lors, lorsque l'événement est émis suite à une action de l'utilisateur, la méthode correspondante du Listener enregistré est exécutée.

Dans la pratique, les Listeners sont des interfaces qu'il faut faire implémenter par une classe selon les besoins. Cette implémentation définira donc des méthodes qui contient les traitements à exécuter pour un événement précis. Un ou plusieurs paramètres fournis à ces méthodes permettent d'obtenir des informations plus précises sur l'événement.

Il suffit ensuite d'enregistrer le Listener auprès du contrôle en utilisant la méthode addXXXListener() du contrôle ou XXX représente le type du Listener.

Exemple : pour le traitement d'un clic d'un bouton

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT3 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    Button button = new Button(shell, SWT.NONE);
    button.setText("Valider");
    button.setBounds(1, 1, 100, 25);

    button.addSelectionListener(new SelectionListener() {
      public void widgetSelected(SelectionEvent arg0) {
                      System.out.println("Appui sur le bouton");
      }
      public void widgetDefaultSelected(SelectionEvent arg0) {
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
```

```
}
```

Comme avec Swing, SWT propose un ensemble de classe de type Adapter qui sont des classes qui implémentent une des interfaces Listeners avec toutes ces méthodes vides. Pour les utiliser, il suffit de définir une classe fille qui hérite de la classe de type Adapter adéquat et de redéfinir la ou les méthodes utiles.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT4 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    Button button = new Button(shell, SWT.NONE);
    button.setText("Valider");
    button.setBounds(1, 1, 100, 25);

    button.addSelectionListener(new SelectionAdapter() {
      public void widgetSelected(SelectionEvent arg0) {
        System.out.println("Appui sur le bouton");
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

SWT définit plusieurs Listeners :

- SelectionListener : événement lié à la sélection d'un élément du contrôle
- KeyListener : événement lié au clavier
- MouseListener : événement lié aux clics de la souris
- MouseMoveListener : événement lié au mouvement de la souris
- MouseTrackListener : événement lié à la souris par rapport au contrôle (entrée, sortie, passage au dessus)
- ModifyListerner : événement lié à la modification du contenu d'un contrôle de saisie de texte
- VerifyListener : événement lié à la vérification avant modification du contenu d'un contrôle de saisie de texte
- FocusListener : événement lié à prise ou à la perte du focus
- TraverseListener : événement lié à la traversée d'un contrôle au moyen de la touche tab ou des flèches
- PaintListener : événement lié à nécessité de redessiner le composant

15.10.1. L'interface KeyListener

Cette interface définit deux méthodes keyPressed() et keyReleased() relatives à des événements émis par le clavier, respectivement l'enfoncement d'une touche et la relache d'une touche du clavier.

Ces deux méthodes possèdent un objet de type KeyEvent qui contient des informations sur l'événements grâce à trois attributs :

character	contient le caractère la touche concernée
keyCode	contient le code de la touche concernée SWT défini des valeurs pour des touches particulières, par exemple SWT.ALT, SWT.ARROW_DOWN, SWT_ARROW_LEFT, SWT.CTRL, SWT.CR, SWT.F1, SWT.F2, ...
stateMask	contient l'état du clavier au moment de l'émission de l'événement, ce qui permet de savoir par exemple si la touche Alt ou Shift ou Ctrl est enfoncée au moment de l'événement en effectuant un test sur la valeur avec SWT.ALT ou SWT.CTRL ou SWT.SHIFT

SWT définit une classe KeyAdapter qui implémente l'interface KeyListener avec des méthodes vides.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT5 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    shell.addKeyListener(new KeyAdapter() {
      public void keyReleased(KeyEvent e) {
        String res = "";
        switch (e.character) {
          case SWT.CR :
            res = "Touche Entree";
            break;
          case SWT.DEL :
            res = "Touche Supp";
            break;
          case SWT.ESC :
            res = "Touche Echap";
            break;
          default :
            res = res + e.character;
        }

        System.out.println(res);

      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

Exemple : utilisation de la propriété stateMask

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT6 {

  public static void main(String[] args) {
    Display display = new Display();
```

```
    Shell shell = new Shell(display);
    shell.setText("Test");

    shell.addKeyListener(new KeyAdapter() {
      public void keyReleased(KeyEvent e) {
        String res = "";
        if (e.keyCode == SWT.SHIFT) {
            res = "touche shift";
        } else {
          if ((e.stateMask & SWT.SHIFT) != 0) {
            res = "" + e.character + " + touche shift";
          }
        }
        System.out.println(res);
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.2. L'interface MouseListener

Cette interface définit trois méthodes mouseDown(), mouseUp() et mouseDoubleClick() relative à des événements émis par un clic sur la souris, respectivement l'enfoncement d'un bouton et le relachement d'un bouton ou le double clic sur un bouton de la souris.

Ces trois méthodes possèdent un objet de type MouseEvent qui contient des informations sur l'événements grâce à quatre attributs :

button	contient le numéro du bouton utilisé (de 1 à 3). Attention, la valeur est dépendante du système utilisé, par exemple sous Windows avec une souris à molette possédant deux boutons, l'appui sur le bouton de droite renvoie 3
stateMask	contient l'état du clavier au moment de l'émission de l'événement, ce qui permet de savoir par exemple si la touche Alt ou Shift ou Ctrl est enfoncée au moment de l'événement en effectuant un test sur la valeur avec SWT.ALT ou SWT.CTRL ou SWT.SHIFT
x	contient la coordonnée x du pointeur de la souris par rapport au contrôle lors de l'émission de l'événement
y	contient la coordonnée y du pointeur de la souris par rapport au contrôle lors de l'émission de l'événement

SWT définit une classe MouseAdapter qui implémente l'interface MouseListener avec des méthodes vides.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT7 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");
```

```
    shell.addMouseListener(new MouseAdapter() {
      public void mouseDown(MouseEvent e) {
        String res = "";
          res = "bouton " + e.button + ", x = " + e.x + ", y = " + e.y;
          if ((e.stateMask & SWT.SHIFT) != 0) {
            res = res + " + touche shift";
          }
        System.out.println(res);
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.3. L'interface MouseMoveListener

Cette interface définit une seule méthode mouseMove() relative à un événement émis par un déplacement de la souris au dessus d'un contrôle.

Cette méthode possède un objet de type MouseEvent qui contient des informations sur l'événement grâce à quatre attributs.

La mise en oeuvre est similaire de l'interface MouseListener.

Exemple :

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT8 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    shell.addMouseMoveListener(new MouseMoveListener() {
      public void mouseMove(MouseEvent e) {
        String res = "";
        if ((e.x < 20) & (e.y < 20)) {

          res = "x = " + e.x + ", y = " + e.y;
          if ((e.stateMask & SWT.SHIFT) != 0) {
            res = res + " + touche shift";
          }
          System.out.println(res);
        }
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
```

```
        display.sleep();

    display.dispose();
  }
}
```

15.10.4. L'interface MouseTrackListener

Cette interface définit trois méthodes mouseEnter(), MouseExit() et mouseHover() relative à des événements émis respectivement par l'entrée de la souris sur la zone d'un composant, la sortie et le passage au dessus de la zone d'un composant.

Ces trois méthodes possèdent un objet de type MouseEvent.

SWT définit une classe MouseTrackAdapter qui implémente l'interface MouseListener avec des méthodes vides.

Exemple : changement de la couleur de fond de la fenêtre

```
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;

public class TestSWT9 {

  public static void main(String[] args) {
    Display display = new Display();
    final Shell shell = new Shell(display);

    final Color couleur1 = new Color(display,155,130,0);
    final Color couleur2 = new Color(display,130,130,130);
    shell.setText("Test");

    shell.addMouseTrackListener(new MouseTrackAdapter() {
      public void mouseEnter(MouseEvent e) {
            shell.setBackground(couleur1);
      }
      public void mouseExit(MouseEvent e) {
            shell.setBackground(couleur2);
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.5. L'interface ModifyListener

Cette interface définit une seule méthode modifyText() relative à un événement émis lors de la modification du contenu d'un contrôle de saisie de texte.

Cette méthode possède un objet de type ModifyEvent.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT10 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    Text text = new Text(shell, SWT.BORDER);
    text.setText("mon texte");
    text.setSize(100, 25);

   text.addModifyListener(new ModifyListener() {
     public void modifyText(ModifyEvent e) {
       System.out.println("nouvelle valeur = " + ((Text)e.widget).getText());
      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.6. L'interface VerifyText()

Cette interface définit une seule méthode verifyText() relative à un événement émis lors de la vérification des données avant modification du contenu d'un contrôle de saisie de texte.

Cette méthode possède un objet de type VerifyEvent qui contient des informations sur l'événement grâce à quatre attributs :

doit	un drapeau qui indique si la modification doit être effectuée ou non
end	la position de début de la modification
start	la position de fin de la modification
text	la valeur de la modification

Exemple : n'autoriser la saisie que de chiffre

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class TestSWT11 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Test");

    Text text = new Text(shell, SWT.BORDER);
    text.setText("");
    text.setSize(100, 25);

    text.addVerifyListener(new VerifyListener() {
      public void verifyText(VerifyEvent e) {
```

```
        int valeur = 0;
        e.doit = true;
        if (e.text != "") {
          try {
            valeur = Integer.parseInt(e.text);
          } catch (NumberFormatException e1) {
            e.doit = false;
          }
        }

        System.out.println(
          "start = " + e.start + ", end = " + e.end + ", text = " + e.text);

      }
    });

    shell.pack();
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.7. L'interface FocusListener

Cette interface définit deux méthodes focusGained() et focusLost() relative à un événement émis respectivement lors de la prise et la perte du focus par un contrôle.

Ces méthodes possèdent un objet de type FocusEvent.

SWT définit une classe FocusAdapter qui implémente l'interface FocusListener avec des méthodes vides.

Exemple :

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.*;

public class TestSWT12 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);

    Text text = new Text(shell, SWT.BORDER);
    text.setText("mon texte");
    text.setBounds(10, 10, 100, 25);

    text.addFocusListener(new FocusListener() {
      public void focusGained(FocusEvent e) {
        System.out.println(e.widget + " obtient le focus");
      }
      public void focusLost(FocusEvent e) {
        System.out.println(e.widget + " perd le focus");
      }
    });

    Button button = new Button(shell, SWT.NONE);
    button.setText("Valider");
    button.setBounds(10, 40, 100, 25);

    shell.pack();
```

```
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.8. L'interface TraverseListener

Cette interface définit une méthode keyTraversed() relative à un événement émis lors de la traversée d'un contrôle au moyen de la touche tab ou des flèches haut et bas.

Cette méthode possède un objet de type VerifyEvent qui contient des informations sur l'événement grâce à deux attributs :

doit	un drapeau qui indique si le composant peut être traversé ou non
detail	le type de l'opération qui génère la traversée

Exemple : empêcher le parcours des contrôles dans l'ordre inverse par la touche tab

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.*;

public class TestSWT13 {

  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);

    TraverseListener tl = new TraverseListener() {
      public void keyTraversed(TraverseEvent e) {
        String res = "";
        res = e.widget + " est traverse grace à ";
        switch (e.detail) {
          case SWT.TRAVERSE_TAB_NEXT :
            res = res + " l'appui sur la touche tab";
            e.doit = true;
            break;
          case SWT.TRAVERSE_TAB_PREVIOUS :
            res = res + " l'appui sur la touche shift + tab";
            e.doit = false;
            break;
          default :
            res = res + " un autre moyen";
        }
        System.out.println(res);
      }
    };

    Text text = new Text(shell, SWT.BORDER);
    text.setText("mon texte");
    text.setBounds(10, 10, 100, 25);
    text.addTraverseListener(tl);

    Button button = new Button(shell, SWT.NONE);
    button.setText("Valider");
    button.setBounds(10, 40, 100, 25);
    button.addTraverseListener(tl);

    shell.pack();
```

```
    shell.open();

    while (!shell.isDisposed())
      if (!display.readAndDispatch())
        display.sleep();

    display.dispose();
  }
}
```

15.10.9. L'interface PaintListener

Cette interface définit une méthode paintControl() relative à un événement émis lors de la nécessité de redessiner le composant.

Cette méthode possède un objet de type PaintEvent qui contient des informations sur l'événement grâce à plusieurs attributs :

gc	un objet de type GC qui encapsule le contexte graphique
height	la hauteur de la zone à redessiner
width	la longueur de la zone à redessiner
x	l'abscisse de la zone à redessiner
y	l'ordonnée de la zone à redessiner

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;

public class TestSWT21 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setSize(420, 420);

    Canvas canvas = new Canvas(shell, SWT.NONE);
    canvas.setSize(200, 200);
    canvas.setLocation(10, 10);
    canvas.addPaintListener(new PaintListener() {
      public void paintControl(PaintEvent e) {
        GC gc = e.gc;
        gc.drawText("Bonjour", 20, 20);
        gc.drawLine(10, 10, 10, 100);
        gc.setForeground(display.getSystemColor(SWT.COLOR_RED));
        gc.drawOval(60, 60, 60, 60);
      }
    });

    shell.pack();
    shell.open();

    GC gc = new GC(canvas);

    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    display.dispose();
```

```
  }

}
```

Remarque : il est important d'associer le listener avant d'ouvrir la fenêtre. Il n'est pas utile d'uiliser la méthode dispose() de l'objet de type GC car le code n'est pas responsable de son instanciation.

15.11. Les boîtes de dialogue

15.11.1. Les boîtes de dialogues prédéfinies

SWT propose plusieurs boîtes de dialogue prédéfinies.

15.11.1.1. La classe MessageBox

La classe MessageBox permet d'afficher un message à l'utilisateur et éventuellement de sélectionner une action standard via un bouton.

Les styles utilisables avec MessageBox sont :

_ ICON_ERROR, ICON_INFORMATION, ICON_QUESTION, ICON_WARNING, ICON_WORKING pour sélectionner l'icône affichée dans la boîte de dialogue

_ OK ou OK | CANCEL : pour boîte avec des boutons de type « Ok » / « Annuler »

_ YES | NO, YES | NO | CANCEL : pour boîte avec des boutons de type « Oui » / « Non » / « Annuler »

_ RETRY | CANCEL : pour boîte avec des boutons de type « Réessayer » / « Annuler »

_ ABORT | RETRY | IGNORE : pour boîte avec des boutons de type « Abandon » / « Réessayer » / « Ignorer »

La méthode setMessage() permet de préciser le message qui va être affiché à l'utilisateur.

La méthode open permet d'ouvrir la boîte de dialogue et de connaître le bouton qui à été utilisé pour fermer la boîte de dialogue en comparant la valeur de retour avec la valeur de style du bouton correspondant.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;

public class TestSWT18 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300, 300);

    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Afficher");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        int reponse = 0;
        MessageBox mb = new MessageBox(shell,
          SWT.ICON_INFORMATION | SWT.ABORT | SWT.RETRY | SWT.IGNORE);
```

```
        mb.setMessage("Message d'information pour l'utilisateur");
        reponse = mb.open();
        if (reponse == SWT.ABORT) {
          System.out.println("Bouton abandonner selectionne");
        }
        if (reponse == SWT.RETRY) {
          System.out.println("Bouton reessayer selectionne");
        }
        if (reponse == SWT.IGNORE) {
          System.out.println("Bouton ignorer selectionne");
        }
      }
    });

    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }

}
```

15.11.1.2. La classe ColorDialog

Cette boîte de dialogue permet la sélection d'une couleur dans la palette des couleurs.

La méthode setRGB() permet de préciser la couleur qui est sélectionnée par défaut.

La méthode open() permet d'ouvrir la boîte de dialogue et de renvoyer la valeur de la couleur sélectionné sous la forme d'un objet de type RGB. Si aucune couleur n'est sélectionnée (appui sur le bouton annuler dans la boîte de dialogue) alors l'objet renvoyé est null.

Exemple :

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.graphics.*;

public class TestSWT15 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300, 300);

    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Couleur");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        Color couleurDeFond = shell.getBackground();

        ColorDialog colorDialog = new ColorDialog(shell);
        colorDialog.setRGB(couleurDeFond.getRGB());
        RGB couleur = colorDialog.open();

        if (couleur != null) {
          if (couleurDeFond != null)
            couleurDeFond.dispose();
          couleurDeFond = new Color(display, couleur);
```

```
          shell.setBackground(couleurDeFond);
        }

      }
    });

    shell.getBackground().dispose();
    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }

}
```

15.11.1.3. La classe FontDialog

Cette classe encapsule une boîte de dialogue permettant la sélection d'une police de caractère.

La méthode open() permet d'ouvrir la boîte de dialogue et renvoie un objet de type FontData qui encapsule les données de la police sélectionnée ou renvoie null si aucune n'a été sélectionnée.

Exemple :

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;

public class TestSWT19 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300, 300);

    Label lblNomPolice = new Label(shell, SWT.NONE);
    lblNomPolice.setText("Nom de la police = ");
    final Text txtNomPolice = new Text(shell, SWT.BORDER | SWT.READ_ONLY);
    txtNomPolice.setText("");
    txtNomPolice.setSize(280, 40);

    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Police");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        FontDialog dialog = new FontDialog(shell, SWT.OPEN);
        FontData fontData = dialog.open();
        if (fontData != null) {
          txtNomPolice.setText(fontData.getName());
          System.out.println("selection de la police " + fontData.getName());
          if (txtNomPolice.getFont() != null) {
            txtNomPolice.getFont().dispose();
          }
          Font font = new Font(display, fontData);
          txtNomPolice.setFont(font);

        }
      }
    });

    shell.pack();
    shell.open();
```

```
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }

}
```

15.11.1.4. La classe FileDialog

Cette boîte de dialogue permet de sélectionner un fichier.

La méthode open() ouvre la boîte de dialogue et renvoie le nom du fichier sélectionné. Si aucun fichier n'est sélectionné, alors elle renvoie null.

La méthode setFilterExtensions() permet de préciser sous la forme d'un tableau de chaîne la liste des extensions de fichier acceptés par la sélection.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;

public class TestSWT16 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300, 300);

    Label lblNomFichier = new Label(shell, SWT.NONE);
    lblNomFichier.setText("Nom du fichier = ");
            final Text txtNomFichier = new Text(shell, SWT.BORDER | SWT.READ_ONLY);
            txtNomFichier.setText("");
            txtNomFichier.setSize(280,40);


    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Ouvrir");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        String nomFichier;
        FileDialog dialog = new FileDialog(shell, SWT.OPEN);
        dialog.setFilterExtensions(new String[] { "*.java", "*.*" });
        nomFichier = dialog.open();
        if ((nomFichier != null) && (nomFichier.length() != 0)){
          txtNomFichier.setText(nomFichier);
            System.out.println("selection du fichier "+nomFichier);
        }
      }
    });

    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }

    display.dispose();
  }

}
```

15.11.1.5. La classe DirectoryDialog

Cette classe encapsule une boîte de dialogue qui permet la sélection d'un répertoire.

La méthode open() ouvre la boîte de dialogue et renvoie le nom du répertoire sélectionné. Si aucun répertoire n'est sélectionné, alors elle renvoie null.

La méthode setFilterPath () permet de préciser sous la forme d'une chaîne de caractère le répertoire sélectionné par défaut.

La méthode setMessage() permet de préciser un message qui sera affiché dans la boîte de dialogue.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;

public class TestSWT17 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300, 300);

    Label lblNomFichier = new Label(shell, SWT.NONE);
    lblNomFichier.setText("Nom du fichier = ");
            final Text txtNomRepertoire = new Text(shell, SWT.BORDER | SWT.READ_ONLY);
            txtNomRepertoire.setText("");
            txtNomRepertoire.setSize(280,40);


    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Ouvrir");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        String nomRepertoire;
        DirectoryDialog dialog = new DirectoryDialog(shell, SWT.OPEN);
        dialog.setFilterPath("d:/");
        dialog.setMessage("Test");
        nomRepertoire = dialog.open();
        if ((nomRepertoire != null) && (nomRepertoire.length() != 0)){
          txtNomRepertoire.setText(nomRepertoire);
          System.out.println("selection du repertoire "+nomRepertoire);
        }
      }
    });

    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }

}
```

15.11.1.6. La classe PrintDialog

La classe PrintDialog encapsule une boîte de dialogue permattant la sélection d'une imprimante configurée sur le système. Pour utiliser classe, il faut importer la package org.eclipse.swt.printing.

La méthode open() permet d'ouvrir la boîte de dialogue et renvoie un objet de type PrinterData qui encapsule les données de l'imprimante sélectionnée ou renvoie null si aucune n'a été sélectionnée.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.printing.*;
import org.eclipse.swt.graphics.*;

public class TestSWT20 {

  public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300, 300);

    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Imprimer");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        PrintDialog dialog = new PrintDialog(shell, SWT.OPEN);
        PrinterData printerData = dialog.open();
        if (printerData != null) {
          Printer printer = new Printer(printerData);
          if (printer.startJob("Test")) {
            printer.startPage();
            GC gc = new GC(printer);
            gc.drawString("Bonjour", 100, 100);
            printer.endPage();
            printer.endJob();
            gc.dispose();
            printer.dispose();
          }
        }
      }
    });

    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }

}
```

Remarque : cet exemple est très basic dans la mesure ou il est préférable de lancer les taches d'impression dans un thread pour ne pas bloquer l'interface utilisateur pendant ces traitements.

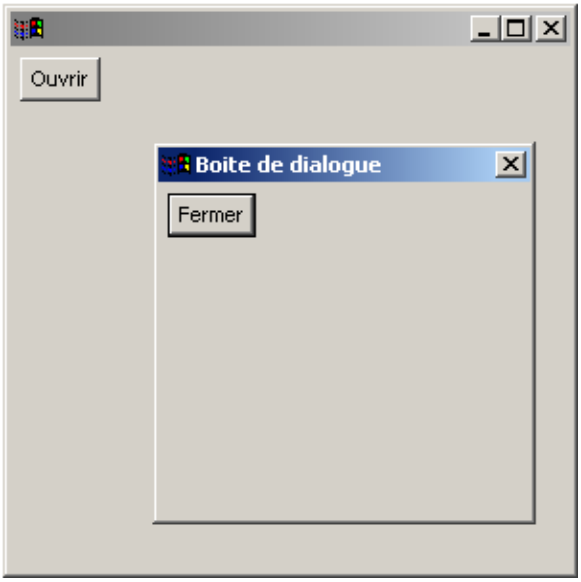
15.11.2. Les boites de dialogues personnalisées

Pour définir une fenêtre qui sera une boite de dialogue, il suffit de définir un nouvel objet de type Shell qui sera lui même rattaché à sa fenêtre père.

Exemple :

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;

public class TestSWT14 {
```

```
  public static void main(String[] args) {
    Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setSize(300,300);

    Button btnOuvrir = new Button(shell, SWT.PUSH);
    btnOuvrir.setText("Ouvrir");
    btnOuvrir.addListener(SWT.Selection, new Listener() {
      public void handleEvent(Event e) {
        final Shell fenetreFille = new Shell(shell, SWT.TITLE | SWT.CLOSE);
        fenetreFille.setText("Boite de dialogue");
                        fenetreFille.setLayout(new GridLayout());

        fenetreFille.addListener(SWT.Close, new Listener() {
          public void handleEvent(Event e) {
            System.out.println("Fermeture de la boite de dialogue");
          }
        });

        Button btnFermer = new Button(fenetreFille, SWT.PUSH);
        btnFermer.setText("Fermer");
        btnFermer.addListener(SWT.Selection, new Listener() {
          public void handleEvent(Event e) {
            fenetreFille.close();
          }
        });
        fenetreFille.setSize(200, 200);
        fenetreFille.open();
      }
    });

    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
  }

}
```

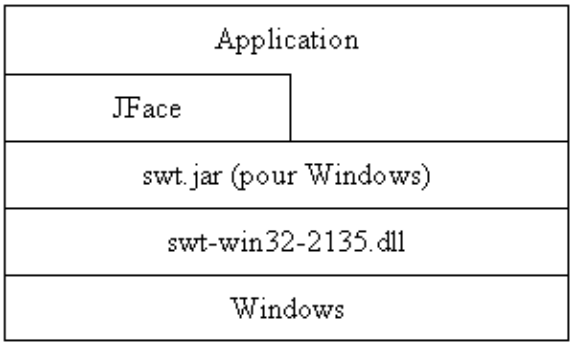


16. JFace

SWT est une API de bas niveau. Elle propose des objets qui permettent la création d'interfaces graphiques mais qui nécessitent aussi énormément de code.

JFace propose d'encapsuler de nombreuses opérations de base et de faciliter ainsi le développement des interfaces graphiques reposant sur SWT.

L'API de JFace est indépendante du système graphique utilisé : la dépendance est réalisée par SWT sur lequel JFace repose.



17. Les applets

Une applet est un programme Java qui s'exécute dans un logiciel de navigation supportant java ou dans l'appletviewer du JDK.

Attention : il est recommandé de tester les applets avec l'appletviewer car les navigateurs peuvent prendre l'applet contenu dans leur cache plutôt que la dernière version compilée.

Le mécanisme d'initialisation d'une applet se fait en deux temps :

1. la machine virtuelle java instancie l'objet Applet en utilisant le constructeur par défaut
2. la machine virtuelle java envoie le message init à l'objet Applet

Ce chapitre contient plusieurs sections :

17.1. L'intégration d'applets dans une page HTML

Dans une page HTML, il faut utiliser le tag APPLET avec la syntaxe suivante :

<APPLET CODE=« Exemple.class » WIDTH=200 HEIGHT=300 > </APPLET>

Le nom de l'applet est indiqué entre guillemets à la suite du parametre CODE.

Les paramètres WIDTH et HEIGHT fixent la taille de la fenêtre de l'applet dans la page HTML. L'unité est le pixel. Il est préférable de ne pas dépasser 640 * 480 (VGA standard).

Le tag APPLET peut comporter les attributs facultatifs suivants :

Tag	Role
CODEBASE	permet de spécifier le chemin relatif par rapport au dossier de la page contenant l'applet. Ce paramètre suit le paramètre CODE. Exemple : CODE=nomApplet.class CODEBASE=/nomDossier
HSPACE et VSPACE	permettent de fixer la distance en pixels entre l'applet et le texte
ALT	

	affiche le texte specifié par le parametre lorsque le navigateur ne supporte pas Java ou que son support est désactivé.

Le tag PARAM permet de passer des paramètres à l'applet. Il doit être inclus entre les tags APPLET et /APPLET.

<PARAM nomParametre value=« valeurParametre »> </APPLET>

La valeur est toujours passée sous forme de chaîne de caractères donc entourée de guillemets.

Exemple : <APPLET code=« Exemple.class » width=200 height=300>

Le texte contenu entre <APPLET> et </APPLET> est afficher si le navigateur ne supporte pas java.

17.2. Les méthodes des applets

Une classe dérivée de la classe java.applet.Applet hérite de méthodes qu'il faut redéfinir en fonction des besoins et doit être déclarée public pour fonctionner.

En général, il n'est pas nécessaire de faire un appel explicite aux méthode init(), start(), stop() et destroy() : le navigateur se charge d'appeler ces méthodes en fonction de l'état de la page HTML contenant l'applet.

17.2.1. La méthode init()

Cette méthode permet l'initialisation de l'applet : elle n'est exécutée qu'une seule et unique fois après le chargement de l'applet.

17.2.2. La méthode start()

Cette méthode est appelée automatiquement après le chargement et l'initialisation (via la méthode init()) lors du premier affichage de l'applet.

17.2.3. La méthode stop()

Le navigateur appelle automatiquement la méthode lorsque l'on quitte la page HTML. Elle interrompt les traitements de tous les processus en cours.

17.2.4. La méthode destroy()

Elle est appelée après l'arrêt de l'applet ou lors de l'arret de la machine virtuelle. Elle libère les ressources et détruit les threads restants

17.2.5. La méthode update()

Elle est appelée à chaque rafraichissement de l'écran ou appel de la méthode repaint(). Elle efface l'écran et appelle la méthode paint(). Ces actions provoquent souvent des scintillements. Il est préférable de redéfinir cette méthode pour qu'elle n'efface plus l'écran :

Exemple :

```
public void update(Graphics g) { paint (g);}
```

17.2.6. La méthode paint()

Cette méthode permet d'afficher le contenu de l'applet à l'écran. Ce refraîchissement peut être provoqué par le navigateur ou par le système d'exploitation si l'ordre des fenêtres ou leur taille ont été modifiés ou si une fenêtre recouvre l'applet.

Exemple :

```
public void paint(Graphics g)
```

La méthode repaint() force l'utilisation de la méthode paint().

Il existe des méthodes dédiées à la gestion de la couleur de fond et de premier plan

La méthode setBackground(Color), héritée de Component, permet de définir la couleur de fond d'une applet. Elle attend en paramètre un objet de la classe Color.

La méthode setForeground(Color) fixe la couleur d'affichage par défaut. Elle s'applique au texte et aux graphiques.

Les couleurs peuvent être spécifiées de trois manières différentes :

utiliser les noms standard predéfinis	Color.nomDeLaCouleur Les noms predéfinis de la classe Color sont : black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow

utiliser 3 nombres de type entier représentant le RGB	(Red,Green,Blue : rouge,vert, bleu)
	
	Exemple :
	`Color macouleur = new Color(150,200,250);`
	`setBackground (macouleur);`
	`// ou setBackground(150,200,250);`

utiliser 3 nombres de type float utilisant le systeme HSB	(Hue, Saturation, Brightness : teinte, saturation, luminance). Ce système est moins répandu que le RGB mais il permet notamment de modifier la luminance sans modifier les autres caractéristiques Exemple : `setBackground(0.0,0.5,1.0);` dans ce cas 0.0,0.0,0.0 représente le noir et 1.0,1.0,1.0 représente le blanc.

17.2.7. Les méthodes size() et getSize()

L'origine des coordonnées en Java est le coin supérieur gauche. Elles s'expriment en pixels avec le type int.

La détermination des dimensions d'une applet se fait de la façon suivante :

Exemple (code Java 1.0) :

```
Dimension dim = size();
int applargeur = dim.width;
```

```
int apphauteur = dim.height;
```

Avec le JDK 1.1, il faut utiliser getSize() à la place de size().

Exemple (code Java 1.1) :

```
public void paint(Graphics g) {
   super.paint(g);
   Dimension dim = getSize();
   int applargeur = dim.width;
   int apphauteur = dim.height;
   g.drawString("width = "+applargeur,10,15);
   g.drawString("height = "+apphauteur,10,30);
}
```

17.2.8. Les méthodes getCodeBase() et getDocumentBase()

Ces méthodes renvoient respectivement l'emplacement de l'applet sous forme d'adresse Web ou de dossier et l'emplacement de la page HTML qui contient l'applet.

Exemple :

```
public void paint(Graphics g) {
   super.paint(g);
   g.drawString("CodeBase = "+getCodeBase(),10,15);
   g.drawString("DocumentBase = "+getDocumentBase(),10,30);
}
```

17.2.9. La méthode showStatus()

Affiche une message dans la barre de status de l'applet

Exemple :

```
public void paint(Graphics g) {
   super.paint(g);
   showStatus("message à afficher dans la barre d'état");
}
```

17.2.10. La méthode getAppletInfo()

Permet de fournir des informations concernant l'auteur, la version et le copyright de l'applet

Exemple :

```
static final String appletInfo = " test applet : auteur, 1999 \n\nCommentaires";

public String getAppletInfo() {
   return appletInfo;
}
```

Pour voir les informations, il faut utiliser l'option info du menu Applet de l'appletviewer.

17.2.11. La méthode getParameterInfo()

Cette méthode permet de fournir des informations sur les paramètres reconnus par l'applet

Le format du tableau est le suivant :

{ {nom du paramètre, valeurs possibles, description} , ... }

Exemple :

```
static final String[][] parameterInfo =
{ {"texte1", "texte1", " commentaires du texte 1" } ,
  {"texte2", "texte2", " commentaires du texte 2" } };

public String[][] getParameterInfo() {
   return parameterInfo;
}
```

Pour voir les informations, il faut utiliser l'option info du menu Applet de l'appletviewer.

17.2.12. La méthode getGraphics()

Elle retourne la zone graphique d'une applet : utile pour dessiner dans l'applet avec des méthodes qui ne possèdent pas le contexte graphique en paramètres (ex : mouseDown ou mouseDrag).

17.2.13. La méthode getAppletContext()

Cette méthode permet l'accès à des fonctionnalités du navigateur.

17.2.14. La méthode setStub()

Cette méthode permet d'attacher l'applet au navigateur.

17.3. Les interfaces utiles pour les applets

17.3.1. L'interface Runnable

Cette interface fournit le comportement nécessaire à un applet pour devenir un thread.

Les méthodes start() et stop() de l'applet peuvent permettre d'arrêter et de démarrer un thread pour permettre de limiter l'usage des ressources machines lorsque la page contenant l'applet est inactive.

17.3.2. L'interface ActionListener

Cette interface permet à l'applet de répondre aux actions de l'utilisateur avec la souris

La méthode actionPerformed() permet de définir les traitements associés aux événements.

Exemple (code Java 1.1) :

```
public void actionPerformed(ActionEvent evt) { ... }
```

Pour plus d'information, voir le chapitre "l'interception des actions de l'utilisateur".

17.3.3. L'interface MouseListener pour répondre à un clic de souris

Exemple (code Java 1.1) :

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class AppletMouse extends Applet implements MouseListener {
   int nbClick = 0;

   public void init() {
      super.init();
      addMouseListener(this);
   }

   public void mouseClicked(MouseEvent e) {
      nbClick++;
      repaint();
   }

   public void mouseEntered(MouseEvent e) {
   }

   public void mouseExited(MouseEvent e) {
   }

   public void mousePressed(MouseEvent e) {
   }

   public void mouseReleased(MouseEvent e) {
   }

   public void paint(Graphics g) {
      super.paint(g);
      g.drawString("Nombre de clics : " + nbClick, 10, 10);
   }

}
```

Pour plus d'information, voir le chapitre sur "l'interception des actions de l'utilisateur".

17.4. La transmission de paramètres à une applet

La méthode getParameter() retourne les paramètres écrits dans la page HTML. Elle retourne une chaîne de caractères de type String.

Exemple :

```
Sting parametre;
parametre = getParameter(« nom-parametre »);
```

Si le paramètre n'est pas renseigné dans la page HTML alors getParameter() retourne null

Pour utiliser les valeurs des paramètres, il sera souvent nécessaire de faire une conversion de la chaîne de caractères dans le type voulu en utilisant les Wrappers

Exemple :

```
String taille;
int hauteur;
taille = getParameter(« hauteur »);
Integer temp = new Integer(taille)
hauteur = temp.intValue();
```

Exemple :

```
int vitesse;
String paramvitesse = getParameter(« VITESSE »);
if (paramvitesse != null) vitesse = Integer.parseInt(paramVitesse);
// parseInt ne fonctionne pas avec une chaîne vide
```

Attention : l'appel à la méthode getParameter() dans le constructeur pas défaut lève une exception de type NullPointerException.

Exemple :

```
public MonApplet() {
   String taille;
   taille = getParameter(" message ");
}
```

17.5. Applet et le multimédia

17.5.1. Insertion d'images.

Java supporte deux standards :

- le format GIF de Compuserve qui est beaucoup utilisé sur internet car il génère des fichiers de petite taille contenant des images d'au plus 256 couleurs.
- et le format JPEG. qui convient mieux aux grandes images et à celles de plus de 256 couleurs car le taux de compression avec perte de qualité peut être précisé.

Pour la manipulation des images, le package nécessaire est java.awt.image.

La méthode getImage() possède deux signatures : getImage(URL url) et getImage (URL url, String name).

On procède en deux étapes : le chargement puis l'affichage. Si les paramètres fournis à getImage ne désignent pas une image, aucune exception n'est levée.

La méthode getImage() ne charge pas de données sur le poste client. Celles ci seront chargées quand l'image sera dessinée pour la première fois.

Exemple :

```
public void paint(Graphics g) {
   super.paint(g);
```

```
   Image image=null;
   image=getImage(getDocumentBase( ), "monimage.gif"); //chargement de l'image
   g.drawImage(image, 40, 70, this);
}
```

Le sixième paramètre de la méthode drawImage() est un objet qui implémente l'interface ImageObserver. ImageObserver est une interface déclarée dans le package java.awt.image qui sert à donner des informations sur le fichier image. Souvent, on indique this à la place de cet argument représentant l'applet elle même. La classe ImageObserver détecte le chargement et la fin de l'affichage d'une image. La classe Applet contient le comportement qui se charge de faire ces actions d'ou le fait de mettre this.

Pour obtenir les dimensions de l'image à afficher on peut utiliser les méthodes getWidth() et getHeight() qui retourne un nombre entier en pixels.

Exemple :

```
int largeur = 0;
int hauteur = 0;
largeur = image.getWidth(this);
hauteur = image.getHeight(this);
```

17.5.2. Insertion de sons

Seul le format d'extension .AU de Sun est supporté par java. Pour utiliser un autre format, il faut le convertir.

La méthode play() permet de jouer un son.

Exemple :

```
import java.net.URL;

   ...
   try {
      play(new URL(getDocumentBase(), « monson.au »));
   } catch (java.net.MalformedURLException e) {}
```

La méthode getDocumentBase() détermine et renvoie l'URL de l'applet.

Ce mode d'exécution n'est valable que si le son n'est à reproduire qu'une seule fois, sinon il faut utiliser l'interface AudioClip.

Avec trois méthodes, l'interface AudioClip facilite l'utilisation des sons :

- public abstract void play() // jouer une seule fois le fichier
- public abstract void loop() // relancer le son jusqu'à interruption par la méthode stop ou la fin de l'applet
- public abstract void stop() // fin de la reproduction du clip audio

Exemple :

```
import java.applet.*;
import java.awt.*;
import java.net.*;

public class AppletMusic extends Applet {
   protected AudioClip aC = null;

   public void init() {
      super.init();
      try {
      AppletContext ac = getAppletContext();
      if (ac != null)
         aC = ac.getAudioClip(new URL(getDocumentBase(), "spacemusic.au"));
```

```
      else
         System.out.println(" fichier son introuvable ");
      }
      catch (MalformedURLException e) {}
      aC.loop();
   }
}
```

Pour utiliser plusieurs sons dans une applet, il suffit de déclarer plusieurs variables AudioClip.

L'objet retourné par la méthode getAudioClip() est un objet qui implémente l'interface AudioClip défini dans la machine virtuelle car il est très dépendant du système de la plate forme d'exécution.

17.5.3. Animation d'un logo

Exemple :

```
import java.applet.*;
import java.awt.*;

public class AppletAnimation extends Applet implements Runnable {
   Thread thread;
   protected Image tabImage[];
   protected int index;

   public void init() {
      super.init();
      //chargement du tableau d'image
      index = 0;
      tabImage = new Image[2];
      for (int i = 0; i < tabImage.length; i++) {
         String fichier = new String("monimage" + (i + 1) + ".gif ");
         tabImage[i] = getImage(getDocumentBase(), fichier);
      }
   }

   public void paint(Graphics g) {
      super.paint(g);
      // affichage de l'image
      g.drawImage(tabImage[index], 10, 10, this);
   }

   public void run() {
      //traitements exécuté par le thread
      while (true) {
         repaint();
         index++;
         if (index >= tabImage.length)
            index = 0;
         try {
            thread.sleep(500);
         }
         catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
   }

   public void start() {
      //demarrage du tread
      if (thread == null) {
         thread = new Thread(this);
         thread.start();
      }
   }

   public void stop() {
      // arret du thread
```

```
      if (thread != null) {
         thread.stop();
         thread = null;
      }
   }

   public void update(Graphics g) {
      //la redéfinition de la méthode permet d'éviter les scintillements
      paint(g);
   }

}
```

La surcharge de la méthode paint() permet d'éviter le scintillement de l'écran du à l'effacement de l'écran et à son rafraichissement. Dans ce cas, seul le rafraichissement est effectué.

17.6. Applet et application (applet pouvant s'exécuter comme application)

Il faut rajouter une classe main à l'applet, définir une fenêtre qui recevra l'affichage de l'applet, appeler les méthodes init() et start() et afficher la fenêtre.

Exemple (code Java 1.1) :

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class AppletApplication extends Applet implements WindowListener {

   public static void main(java.lang.String[] args) {
      AppletApplication applet = new AppletApplication();
      Frame frame = new Frame("Applet");
      frame.addWindowListener(applet);
      frame.add("Center", applet);
      frame.setSize(350, 250);
      frame.show();
      applet.init();
      applet.start();
   }

   public void paint(Graphics g) {
      super.paint(g);
      g.drawString("Bonjour", 10, 10);
   }

   public void windowActivated(WindowEvent e) { }

   public void windowClosed(WindowEvent e) { }

   public void windowClosing(WindowEvent e) {
      System.exit(0);
   }

   public void windowDeactivated(WindowEvent e) { }

   public void windowDeiconified(WindowEvent e) { }

   public void windowIconified(WindowEvent e) { }

   public void windowOpened(WindowEvent e) { }

}
```

17.7. Les droits des applets

Une applet est une application Java hébergée sur une machine distante (un serveur Web) et qui s'exécute, après chargement, sur la machine client équipée d'un navigateur. Ce navigateur contrôle les accès de l'applet aux ressources locales et ne les autorisent pas systématiquement : chaque navigateur définit sa propre règle.

Le modèle classique de sécurité pour l 'exécution des applets, recommandé par Sun, distingue deux types d'applets : les applets non dignes de confiance (untrusted) qui n'ont pas accès aux ressources locales et externes, les applets dignes de confiance (trusted) qui ont l'accès. Dans ce modèle, une applet est par défaut untrusted.

La signature d'une applet permet de désigner son auteur et de garantir que le code chargé par le client est bien celui demandé au serveur. Cependant, une applet signée n'est pas forcement digne de confiance.

La suite de ce chapitre sera développée dans une version future de ce document

Partie 3 : Les API avancées

Le JDK fourni un certain nombres d'API intégrés au JDK pour des fonctionnalités avancées.

Cette partie contient les chapitres suivants :

- Les collections : propose une revue des classes fournies par le JDK pour gérer des ensembles d'objets
- Les flux : explore les classes utiles à la mise en oeuvre d'un des mécanismes de base pour échanger des données
- La sérialisation : ce procédé permet de rendre un objet persistant
- L'interaction avec le réseau : propose un aperçu des API fournies par Java pour utiliser les fonctionnalités du réseau dans les applications
- L'accès aux bases de données : JDBC : indique comment utiliser JDBC pour accéder aux bases de données
- La gestion dynamique des objets et l'introspection : ces mécanismes permettent dynamiquement de connaître le contenu d'une classe et de l'utiliser
- L'appel de méthodes distantes : RMI : étudie la mise en oeuvre de la technologie RMI pour permettre l'appel de méthodes distantes
- L'internationalisation : traite d'une façon pratique de la possibilité d'internationnaliser une application
- Les composants java beans : examine comment développer et utiliser des composants réutilisables
- Logging : indique comment mettre en oeuvre deux API pour la gestion des logs : Log4J du projet open source jakarta et l'API logging du JDK 1.4
- La sécurité : partie intégrante de Java, elle revêt de nombreux aspects dans les spécifications, la gestion des droits d'exécution et plusieurs API dédiées
- Java Web Start (JWS) : est une technologie qui permet le déploiement d'applications clientes riches à travers le réseau via un navigateur
- JNI (Java Native Interface) : technologie qui permet d'utiliser du code natif dans une classe Java et vice et versa
- JDO (Java Data Object) : API qui standardise et automatise le mapping en des objets Java et un système de gestion de données

- D'autres solutions de mapping objet−relationnel : présente d'autres solutions de mapping open source notamment Hibernate
- Java et XML : présente XML qui est un technologie tendant à s'imposer pour les échanges de données et explore les API java pour utiliser XML

18. Les collections

Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc. ...

Chaque objet contenu dans une collection est appelé un élément.

Ce chapitre contient plusieurs sections :

18.1. Présentation du framework collection

Dans la version 1 du J.D.K., il n'existe qu'un nombre restreint de classes pour gérer des ensembles de données :

- Vector
- Stack
- Hashtable
- Bitset

L'interface Enumeration permet de parcourir le contenu de ces objets.

Pour combler le manque d'objets adaptés, la version 2 du J.D.K. apporte un framework complet pour gérer les collections. Cette bibliothèque contient un ensemble de classes et interfaces. Elle fourni également un certain nombre de classes abstraites qui implémentent partiellement certaines interfaces.

Les interfaces à utiliser par des objets qui gèrent des collections sont :

- Collection : interface qui est implementée par la plupart des objets qui gèrent des collections
- Map : interface qui définit des méthodes pour des objets qui gèrent des collections sous la forme clé/valeur
- Set : interface pour des objets qui n'autorisent pas la gestion des doublons dans l'ensemble
- List : interface pour des objets qui autorisent la gestion des doublons et un accès direct à un élément
- SortedSet : interface qui étend l'interface Set et permet d'ordonner l'ensemble
- SortedMap : interface qui étend l'interface Map et permet d'ordonner l'ensemble

Certaines méthodes définies dans ces interfaces sont dites optionnelles : leur définition est donc obligatoire mais si l'opération n'est pas supportée alors la méthode doit lever une exception particulière. Ceci permet de réduire le nombre d'interfaces et de répondre au maximum de cas.

Le framework propose plusieurs objets qui implémentent ces interfaces et qui peuvent être directement utilisés :

- HashSet : HashTable qui implémente l'interface Set
- TreeSet : arbre qui implémente l'interface SortedSet
- ArrayList : tableau dynamique qui implémente l'interface List
- LinkedList : liste doublement chaînée (parcours de la liste dans les deux sens) qui implémente l'interface List
- HashMap : HashTable qui implémente l'interface Map
- TreeMap : arbre qui implémente l'interface SortedMap

Le framework définit aussi des interfaces pour faciliter le parcours des collections et leur tri :

- Iterator : interface pour le parcours des collections
- ListIterator : interface pour le parcours des listes dans les deux sens et modifier les éléments lors de ce parcours
- Comparable : interface pour définir un ordre de tri naturel pour un objet
- Comparator : interface pour définir un ordre de tri quelconque

Deux classes existantes dans les précédentes versions du JDK ont été modifiées pour implémenter certaines interfaces du framework :

- Vector : tableau à taille variable qui implémente maintenant l'interface List
- HashTable : table de hashage qui implémente maintenant l'interface Map

Le framework propose la classe Collections qui contient de nombreuses méthodes statiques pour réaliser certaines opérations sur une collection. Plusieurs méthodes unmodifiableXXX() (ou XXX représente une interface d'une collection) permettent de rendre une collection non modifiable. Plusieurs méthodes synchronizedXXX() permettent d'obtenir une version synchronisée d'une collection pouvant ainsi être manipulée de façon sûre par plusieurs threads. Enfin plusieurs méthodes permettent de réaliser des traitements sur la collection : tri et duplication d'une liste, recherche du plus petit et du plus grand élément, etc. ...

Le framework fourni plusieurs classes abstraites qui proposent une implémentation partielle d'une interface pour faciliter la création d'un collection personnalisée : AbstractCollection, AbstractList, AbstractMap, AbstractSequentialList et AbstractSet.

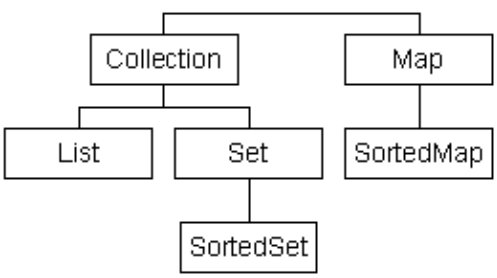
Les objets du framework stockent toujours des références sur les objets contenus dans la collection et non les objets eux mêmes. Ce sont obligatoirement des objets qui doivent être ajoutés dans une collection. Il n'est pas possible de stocker directement des types primitifs : il faut obligatoirement encapsuler ces données dans des wrappers.

Toutes les classes de gestion de collection du framework ne sont pas synchronisées : elles ne prennent pas en charge les traitements multi−threads. Le framework propose des méthodes pour obtenir des objets de gestion de collections qui prennent en charge cette fonctionnalité. Les classes Vector et Hashtable étaient synchronisées mais l'utilisation d'une collection ne se fait généralement pas de ce contexte. Pour réduire les temps de traitement dans la plupart des cas, elles ne sont pas synchronisées par défaut.

Lors de l'utilisation de ces classes, il est préférable de stocker la référence de ces objets sous la forme d'une interface qu'ils implémentent plutôt que sous leur forme objet. Ceci rend le code plus facile à modifier si le type de l'objet qui gèrent la collection doit être changé.

18.2. Les interfaces des collections

Le framework de java 2 définit 6 interfaces en relation directe avec les collections qui sont regroupées dans deux arborescences :



Le JDK ne fourni pas de classes qui implémentent directement l'interface Collection.

Le tableau ci dessous présente les différentes classes qui implémentent les interfaces de bases Set, List et Map :

	Set collection d'éléments uniques	List collection avec doublons	Map collection sous la forme clé/valeur
Tableau redimensionnable		ArrayList,Vector (JDK 1.1)	
Arbre	TreeSet		TreeMap
Liste chaînée		LinkedList	
Collection utilisant une table de hashage	HashSet		HashMap, HashTable (JDK 1.1)
Classes du JDK 1.1		Stack	

Pour gérer toutes les situations de façon simple, certaines méthodes peuvent être définies dans une interface comme «optionnelles ». Pour celles ci, les classes qui implémentent une telle interface, ne sont pas obligées d'implémenter du code qui réalise un traitement mais simplement lève une exception si cette fonctionnalité n'est pas supportée.

Le nombre d'interfaces est ainsi grandement réduit.

Cette exception est du type UnsupportedOperationException. Pour éviter de protéger tous les appels de méthodes d'un objet gérant les collections dans un bloc try−catch, cette exception hérite de la classe RuntimeException.

Toutes les classes fournies par le J.D.K. qui implémentent une des interfaces héritant de Collection implémentent toutes les opérations optionnelles.

18.2.1. L'interface Collection

Cette interface définit des méthodes pour des objets qui gèrent des éléments d'une façon assez générale. Elle est la super interface de plusieurs interfaces du framework.

Plusieurs classes qui gèrent une collection implémentent une interface qui hérite de l'interface Collection. Cette interface est une des deux racines de l'arborescence des collections.

Cette interface définit plusieurs méthodes :

Méthode	Rôle
boolean add(Object)	ajoute l'élément fourni en paramètre à la collection. La valeur de retour indique si la collection a été mise à jour
boolean addAll(Collection)	ajoute à la collection tous les éléments de la collection fournie en paramètre
void clear()	supprime tous les éléments de la collection
boolean contains(Object)	indique si la collection contient au moins un élément identique à celui fourni en paramètre
boolean containsAll(Collection)	indique si tous les éléments de la collection fournie en paramètre sont contenus dans la collection
boolean isEmpty()	indique si la collection est vide
Iterator iterator()	renvoie un objet qui permet de parcourir l'ensemble des éléments de la collection
boolean remove(Object)	supprime l'élément fourni en paramètre de la collection. La valeur de retour indique si la collection a été mise à jour

boolean removeAll(Collection)	supprime tous les éléments de la collection qui sont contenus dans la collection fournie en paramètre
int size()	renvoie le nombre d'éléments contenu dans la collection
Object[] toArray()	renvoie d'un tableau d'objets qui contient tous les éléments de la collection

Cette interface représente un minimum commun pour les objets qui gèrent des collections : ajout d'éléments, suppression d'éléments, vérifier la présence d'un objet dans la collection, parcours de la collection et quelques opérations diverses sur la totalité de la collection.

Ce tronc commun permet entre autre de définir pour chaque objet gérant une collection, un constructeur pour cette objet demandant un objet de type Collection en paramètre. La collection est ainsi initialisée avec les éléments contenus dans la collection fournie en paramètre.

Attention : il ne faut pas ajouter dans une collection une référence à la collection elle−même.

18.2.2. L'interface Iterator

Cette interface définit des méthodes pour des objets capables de parcourir les données d'une collection.

La définition de cette nouvelle interface par rapport à l'interface Enumeration a été justifiée par l'ajout de la fonctionnalité de suppression et la réduction des noms de méthodes.

Méthode	Rôle
boolean hasNext()	indique si il reste au moins à parcourir dans la collection
Object next()	renvoie la prochain élément dans la collection
void remove()	supprime le dernier élément parcouru

La méthode hasNext() est équivalente à la méthode hasMoreElements() de l'interface Enumeration.

La méthode next() est équivalente à la méthode nextElement() de l'interface Enumeration.

La méthode next() lève une exception de type NoSuchElementException si elle est appelée alors que la fin du parcours des éléments est atteinte. Pour éviter la levée de cette exception, il suffit d'appeler la méthode hasNext() et de conditionner avec le résultat l'appel à la méthode next().

Exemple (code Java 1.2) :

```
Iterator iterator = collection.iterator();
while (iterator.hasNext()) {
            System.out.println("objet = "+iterator.next());
}
```

La méthode remove() permet de supprimer l'élément renvoyé par le dernier appel à la méthode next(). Il est ainsi impossible d'appeler la méthode remove() sans un appel correspondant à next() : on ne peut pas appeler deux fois de suite la méthode remove().

Exemple (code Java 1.2) : suppression du premier élément

```
    Iterator iterator = collection.iterator();
     if (iterator.hasNext()) {
           iterator.next();
           itérator.remove();
     }
```

Si aucun appel à la méthode next() ne correspond à celui de la méthode remove(), une exception de type IllegalStateException est levée

18.3. Les listes

Une liste est une collection ordonnée d'éléments qui autorise d'avoir des doublons. Etant ordonné, un élément d'une liste peut être accédé à partir de son index.

18.3.1. L'interface List

Cette interface étend l'interface Collection.

Les collections qui implémentent cette interface autorisent les doublons dans les éléments de la liste. Ils autorisent aussi l'insertion d'éléments null.

L'interface List propose plusieurs méthodes pour un accès à partir d'un index aux éléments de la liste. La gestion de cet index commence à zéro.

Pour les listes, une interface particulière est définie pour assurer le parcours dans les deux sens de la liste et assurer des mises à jour : l'interface ListIterator

Méthode	Rôle
ListIterator listIterator()	renvoie un objet capable de parcourir la liste
Object set (int, Object)	remplace l'élément contenu à la position précisée par l'objet fourni en paramètre
void add(int, Object)	ajouter l'élément fourni en paramètre à la position précisée
Object get(int)	renvoie l'élément à la position précisée
int indexOf(Object)	renvoie l'index du premier élément fourni en paramètre dans la liste ou −1 si l'élément n'est pas dans la liste
ListIterator listIterator()	renvoie un objet pour parcourir la liste et la mettre à jour
List subList(int,int)	renvoie un extrait de la liste contenant les éléments entre les deux index fournis (le premier index est inclus et le second est exclus). Les éléments contenus dans la liste de retour sont des références sur la liste originale. Des mises à jour de ces éléments impactent la liste originale.
int lastIndexOf(Object)	renvoie l'index du dernier élément fourni en paramètre dans la liste ou −1 si l'élément n'est pas dans la liste
Object set(int, Object)	remplace l'élément à la position indiquée avec l'objet fourni

Le framework propose des classes qui implémentent l'interface List : LinkedList et ArrayList.

18.3.2. Les listes chaînées : la classe LinkedList

Cette classe hérite de la classe AbstractSequentialList et implémente donc l'interface List.

Elle représente une liste doublement chaînée.

Cette classe possède un constructeur sans paramètre et un qui demande une collection. Dans ce dernier cas, la liste sera initialisée avec les éléments de la collection fournie en paramètre.

Exemple (code Java 1.2) :

```
LinkedList listeChainee = new LinkedList();
 Iterator iterator = listeChainee.iterator();
 listeChainee.add("element 1");
 listeChainee.add("element 2");
 listeChainee.add("element 3");
 while (iterator.hasNext()) {
       System.out.println("objet = "+iterator.next());
 }
```

Une liste chaînée gère une collection de façon ordonnée : l'ajout d'un élément peut se faire à la fin de la collection ou après n'importe quel élément. Dans ce cas, l'ajout est lié à la position courante lors d'un parcours.

Pour répondre à ce besoin, l'interface qui permet le parcours de la collection est une sous classe de l'interface Iterator : l'interface ListIterator.

Comme les iterator sont utilisés pour faire des mises à jour dans la liste, une exception de type CurrentModificationException levée si un iterator parcours la liste alors qu'un autre fait des mises à jour (ajout ou suppression d'un élément dans la liste).

Pour gérer facilement cette situation, il est préférable si l'on sait qu'il y a des mises à jour à faire de n'avoir qu'un seul iterator qui soit utilisé.

Plusieurs méthodes pour ajouter, supprimer ou obtenir le premier ou le dernier élément de la liste permettent d'utiliser cette classe pour gérer une pile :

Méthode	Rôle
void addFirst(Object)	insère l'objet en début de la liste
void addLast(Object)	insère l'objet en fin de la liste
Object getFirst()	renvoie le premier élément de la liste
Object getLast()	renvoie le dernier élément de la liste
Object removeFirst()	supprime le premier élément de la liste et renvoie le premier élément
Object removeLast()	supprime le dernier élément de la liste et renvoie le premier élément

De par les caractéristiques d'une liste chaînée, il n'existe pas de moyen d'obtenir un élément de la liste directement. Pourtant, la méthode contains() permet de savoir si un élément est contenu dans la liste et la méthode get() permet d'obtenir l'élément à la position fournie en paramètre. Il ne faut toutefois pas oublier que ces méthodes parcourent la liste jusqu'à obtention du résultat, ce qui peut être particulièrement gourmand en terme de temps de réponse surtout si la méthode get() est appelée dans une boucle.

Pour cette raison, il ne faut surtout pas utiliser la méthode get() pour parcourir la liste.

La méthode toString() renvoie une chaîne qui contient tous les éléments de la liste.

18.3.3. L'interface ListIterator

Cette interface définit des méthodes pour parcourir la liste dans les deux sens et effectuer des mises à jour qui agissent par rapport à l'élément courant dans le parcours.

En plus des méthodes définies dans l'interface Iterator dont elle hérite, elle définit les méthodes suivantes :

Méthode	Roles

void add(Object)	ajoute un élément dans la liste en tenant de la position dans le parcours
boolean hasPrevious()	indique si il reste au moins un élément à parcourir dans la liste dans son sens inverse
Object previous()	renvoi l'élément précédent dans la liste
void set(Object)	remplace l'élément courante par celui fourni en paramètre

La méthode add() de cette interface ne retourne pas un booléen indiquant que l'ajout à réussi.

Pour ajouter un élément en début de liste, il suffit d'appeler la méthode add() sans avoir appelé une seule fois la méthode next(). Pour ajouter un élément en fin de la liste, il suffit d'appeler la méthode next() autant de fois que nécessaire pour atteindre la fin de la liste et appeler la méthode add(). Plusieurs appels à la méthode add() successifs, ajoutent les éléments à la position courante dans l'ordre d'appel de la méthode add().

18.3.4. Les tableaux redimensionnables : la classe ArrayList

Cette classe représente un tableau d'objets dont la taille est dynamique.

Elle hérite de la classe AbstractList donc elle implémente l'interface List.

Le fonctionnement de cette classe est identique à celui de la classe Vector.

La différence avec la classe Vector est que cette dernière est multi thread (toutes ces méthodes sont synchronisées). Pour une utilisation dans un thread unique, la synchronisation des méthodes est inutile et coûteuse. Il est alors préférable d'utiliser un objet de la classe ArrayList.

Elle définit plusieurs méthodes dont les principales sont :

Méthode	Rôle
boolean add(Object)	ajoute un élément à la fin du tableau
boolean addAll(Collection)	ajoute tous les éléments de la collection fournie en paramètre à la fin du tableau
boolean addAll(int, Collection)	ajoute tous les éléments de la collection fournie en paramètre dans la collection à partir de la position précisée
void clear()	supprime tous les éléments du tableau
void ensureCapacity(int)	permet d'augmenter la capacité du tableau pour s'assurer qu'il puisse contenir le nombre d'éléments passé en paramètre
Object get(index)	renvoie l'élément du tableau dont la position est précisée
int indexOf(Object)	renvoie la position de la première occurrence de l'élément fourni en paramètre
boolean isEmpty()	indique si le tableau est vide
int lastIndexOf(Object)	renvoie la position de la dernière occurrence de l'élément fourni en paramètre
Object remove(int)	supprime dans le tableau l'élément fourni en paramètre
void removeRange(int,int)	supprime tous les éléments du tableau de la première position fourni incluse jusqu'à la dernière position fournie exclue
Object set(int, Object)	remplace l'élément à la position indiquée par celui fourni en paramètre
int size()	renvoie le nombre d'élément du tableau
void trimToSize()	ajuste la capacité du tableau sur sa taille actuelle

Chaque objet de type ArrayList gère une capacité qui est le nombre total d'élément qu'il est possible d'insérer avant d'agrandir le tableau. Cette capacité a donc une relation avec le nombre d'éléments contenus dans la collection. Lors d'un ajout dans la collection, cette capacité et le nombre d'éléments de la collection déterminent si le tableau doit être agrandi. Si un nombre important d'élément doit être ajouté, il est possible de forcer l'agrandissement de cette capacité avec la méthode ensureCapacity(). Son usage évite une perte de temps liée au recalcul de la taille de la collection. Un constructeur permet de préciser la capacité initiale.

18.4. Les ensembles

Un ensemble (Set) est une collection qui n'autorise pas l'insertion de doublons.

18.4.1. L'interface Set

Cette classe définit les méthodes d'une collection qui n'accepte pas de doublons dans ces éléments. Elle hérite de l'interface Collection mais elle ne définie pas de nouvelle méthode.

Pour déterminer si un élément est déjà inséré dans la collection, la méthode equals() est utilisée.

Le framework propose deux classes qui implémentent l'interface Set : TreeSet et HashSet

Le choix entre ces deux objets est lié à la nécessité de trier les éléments :

- les éléments d'un objet HashSet ne sont pas triés : l'insertion d'un nouvel élément est rapide
- les éléments d'un objet TreeSet sont triés : l'insertion d'un nouvel élément est plus long

18.4.2. L'interface SortedSet

Cette interface définit une collection de type ensemble triée. Elle hérite de l'interface Set.

Le tri de l'ensemble peut être assuré par deux façons :

- les éléments contenus dans l'ensemble implémentent l'interface Comparable pour définir leur ordre naturel
- il faut fournir au constructeur de l'ensemble un objet Comparator qui définit l'ordre de tri à utiliser

Elle définit plusieurs méthodes pour tirer parti de cette ordre :

Méthode	Rôle
Comparator comparator()	renvoie l'objet qui permet de trier l'ensemble
Object first()	renvoie le premier élément de l'ensemble
SortedSet headSet(Object)	renvoie un sous ensemble contenant tous les éléments inférieurs à celui fourni en paramètre
Object last()	renvoie le dernier élément de l'ensemble
SortedSet subSet(Object, Object)	renvoie un sous ensemble contenant les éléments compris entre le premier paramètre inclus et le second exclus
SortedSet tailSet(Object)	renvoie un sous ensemble contenant tous les éléments supérieurs ou égaux à celui fourni en paramètre

18.4.3. La classe HashSet

Cette classe est un ensemble sans ordre de tri particulier.

Les éléments sont stockés dans une table de hashage : cette table possède une capacité.

Exemple (code Java 1.2) :

```
import java.util.*;
public class TestHashSet {
  public static void main(String args[]) {
    Set set = new HashSet();
    set.add("CCCCC");
    set.add("BBBBB");
    set.add("DDDDD");
    set.add("BBBBB");
    set.add("AAAAA");

    Iterator iterator = set.iterator();
    while (iterator.hasNext()) {System.out.println(iterator.next());}
  }
}
```

Résultat :

```
AAAAA
DDDDD
BBBBB
CCCCC
```

18.4.4. La classe TreeSet

Cette classe est un arbre qui représente un ensemble trié d'éléments.

Cette classe permet d'insérer des éléments dans n'importe quel ordre et de restituer ces éléments dans un ordre précis lors de son parcours.

L'implémentation de cette classe insère un nouvel élément dans l'arbre à la position correspondant à celle déterminée par l'ordre de tri. L'insertion d'un nouvel élément dans un objet de la classe TreeSet est donc plus lent mais le tri est directement effectué.

L'ordre utilisé est celui indiqué par les objets insérés si ils implémentent l'interface Comparable pour un ordre de tri naturel ou fournir un objet de type Comparator au constructeur de l'objet TreeSet pour définir l'ordre de tri.

Exemple (code Java 1.2) :

```
import java.util.*;

public class TestTreeSet {
  public static void main(String args[]) {
    Set set = new TreeSet();
    set.add("CCCCC");
    set.add("BBBBB");
    set.add("DDDDD");
    set.add("BBBBB");
    set.add("AAAAA");

    Iterator iterator = set.iterator();
    while (iterator.hasNext()) {System.out.println(iterator.next());}
  }
}
```

Résultat :

```
AAAAA
BBBBB
CCCCC
DDDDD
```

18.5. Les collections gérées sous la forme clé/valeur

Ce type de collection gère les éléments avec deux entités : une clé et une valeur associée. La clé doit être unique donc il ne peut y avoir de doublons. En revanche la même valeur peut être associée à plusieurs clés différentes.

Avant l'apparition du framework collections, la classe dédiée à cette gestion était la classe Hashtable.

18.5.1. L'interface Map

Cette interface est une des deux racines de l'arborescence des collections. Les collections qui implémentent cette interface ne peuvent contenir des doublons. Les collections qui implémentent cette interface utilisent une association entre une clé et une valeur.

Elle définit plusieurs méthodes pour agir sur la collection :

Méthode	Rôle
void clear()	supprime tous les éléments de la collection
boolean containsKey(Object)	indique si la clé est contenue dans la collection
boolean containsValue(Object)	indique si la valeur est contenue dans la collection
Set entrySet()	renvoie un ensemble contenant les valeurs de la collection
Object get(Object)	renvoie la valeur associée à la clé fournie en paramètre
boolean isEmpty()	indique si la collection est vide
Set keySet()	renvoie un ensemble contenant les clés de la collection
Object put(Object, Object)	insère la clé et sa valeur associée fournies en paramètres
void putAll(Map)	insère toutes les clés/valeurs de l'objet fourni en paramètre
Collection values()	renvoie une collection qui contient toutes les éléments des éléments
Object remove(Object)	supprime l'élément dont la clé est fournie en paramètre
int size()	renvoie le nombre d'éléments de la collection

La méthode entrySet() permet d'obtenir un ensemble contenant toutes les clés.

La méthode values() permet d'obtenir une collection contenant toutes les valeurs. La valeur de retour est une Collection et non un ensemble car il peut y avoir des doublons (plusieurs clés peuvent être associées à la même valeur).

Le J.D.K. 1.2 propose deux nouvelles classes qui implémentent cette interface :

- HashMap qui stocke les éléments dans une table de hashage
- TreeMap qui stocke les éléments dans un arbre

La classe HashTable a été mise à jour pour implémenter aussi cette interface.

18.5.2. L'interface SortedMap

Cette interface définit une collection de type Map triée sur la clé. Elle hérite de l'interface Map.

Le tri peut être assuré par deux façons :

- les clés contenues dans la collection implémentent l'interface Comparable pour définir leur ordre naturel
- il faut fournir au constructeur de la collection un objet Comparator qui définit l'ordre de tri à utiliser

Elle définit plusieurs méthodes pour tirer parti de cette ordre :

Méthode	Rôle
Comparator comparator()	renvoie l'objet qui permet de trier la collection
Object first()	renvoie le premier élément de la collection
SortedSet headMap(Object)	renvoie une sous collection contenant tous les éléments inférieurs à celui fourni en paramètre
Object last()	renvoie le dernier élément de la collection
SortedMap subMap(Object, Object)	renvoie une sous collection contenant les éléments compris entre le premier paramètre inclus et le second exclus
SortedMap tailMap(Object)	renvoie une sous collection contenant tous les éléments supérieurs ou égaux à celui fourni en paramètre

18.5.3. La classe Hashtable

Cette classe qui existe depuis le premier jdk implémente une table de hachage. La clé et la valeur de chaque élément de la collection peut être n'importe quel objet non nul.

A partir de Java 1.2 cette classe implémente l'interface Map.

Une des particularités de classe HashTable est qu'elle est synchronisée.

Exemple (code Java 1.2) :

```
import java.util.*;

public class TestHashtable {

  public static void main(String[] args) {

    Hashtable  htable = new Hashtable();
    htable.put(new Integer(3), "données 3");
    htable.put(new Integer(1), "données 1");
    htable.put(new Integer(2), "données 2");

    System.out.println(htable.get(new Integer(2)));

  }
}
```

Résultat :

```
données 2
```

18.5.4. La classe TreeMap

Cette classe gère une collection d'objets sous la forme clé/valeur stockés dans un arbre de type rouge−noir (Red−black tree). Elle implémente l'interface SortedMap. L'ordre des éléments de la collection est maintenu grace à un objet de type Comparable.

Elle possède plusieurs constructeurs dont un qui permet de préciser l'objet Comparable pour définir l'ordre dans la collection.

Exemple (code Java 1.2) :

```
import java.util.*;

public class TestTreeMap {

  public static void main(String[] args) {

    TreeMap arbre = new TreeMap();
    arbre.put(new Integer(3), "données 3");
    arbre.put(new Integer(1), "données 1");
    arbre.put(new Integer(2), "données 2");

    Set cles = arbre.keySet();
    Iterator iterator = cles.iterator();
    while (iterator.hasNext()) {
      System.out.println(arbre.get(iterator.next()));
    }
  }
}
```

Résultat :

```
données 1
données 2
données 3
```

18.5.5. La classe HashMap

La classe HashMap est similaire à la classe Hashtable. Les trois grandes différences sont :

- elle est apparue dans le JDK 1.2
- elle n'est pas synchronisée
- elle autorise les objets null comme clé ou valeur

Cette classe n'étant pas synchonisée, pour assurer la gestion des accès concurents sur cet objet, il faut l'envelopper dans un objet Map en utilisant la méthode synchronizedMap de la classe Collection.

18.6. Le tri des collections

L'ordre de tri est défini grace à deux interfaces :

- Comparable
- Comparator

18.6.1. L'interface Comparable

Tous les objets qui doivent définir un ordre naturel utilisé par le tri d'une collection avec cet ordre doivent implémenter cette interface.

Cette interface ne définit qu'une seule méthode : int compareTo(Object).

Cette méthode doit renvoyer :

- une valeur entière négative si l'objet courant est inférieur à l'objet fourni
- une valeur entière positive si l'objet courant est supérieur à l'objet fourni
- une valeur nulle si l'objet courant est égal à l'objet fourni

Les classes wrappers, String et Date implémentent cette interface.

18.6.2. L'interface Comparator

Cette interface représente un ordre de tri quelconque. Elle est utile pour permettre le tri d'objet qui n'implémente pas l'interface Comparable ou pour définir un ordre de tri différent de celui défini avec Comparable (l'interface Comparable représente un ordre naturel : il ne peut y en avoir qu'un)

Cette interface ne définit qu'une seule méthode : int compare(Object, Object).

Cette méthode compare les deux objets fournis en paramètre et renvoie :

- une valeur entière négative si le premier objet est inférieur au second
- une valeur entière positive si le premier objet est supérieur au second
- une valeur nulle si les deux objets sont égaux

18.7. Les algorithmes

La classe Collections propose plusieurs méthodes statiques qui effectuer des opérations sur des collections. Ces traitements sont polymorphiques car ils demandent en paramètre un objet qui implémente une interface et retourne une collection.

Méthode	Rôle
void copy(List, List)	copie tous les éléments de la seconde liste dans la première
Enumaration enumeration(Collection)	renvoie un objet Enumeration pour parcourir la collection
Object max(Collection)	renvoie le plus grand élément de la collection selon l'ordre naturel des éléments
Object max(Collection, Comparator)	renvoie le plus grand élément de la collection selon l'ordre naturel précisé par l'objet Comparator
Object min(Collection)	renvoie le plus petit élément de la collection selon l'ordre naturel des éléments
Object min(Collection, Comparator)	renvoie le plus petit élément de la collection selon l'ordre précisé par l'objet Comparator
void reverse(List)	inverse l'ordre de la liste fournie en paramètre
void shuffle(List)	réordonne tous les éléments de la liste de façon aléatoire
void sort(List)	trie la liste dans un ordre ascendant selon l'ordre naturel des éléments
void sort(List, Comparator)	trie la liste dans un ordre ascendant selon l'ordre précisé par l'objet Comparator

Si la méthode sort(List) est utilisée, il faut obligatoirement que les éléments inclus dans la liste implémentent tous l'interface Comparable sinon une exception de type ClassCastException est levée.

Cette classe propose aussi plusieurs méthodes pour obtenir une version multi−thread ou non modifiable des principales interfaces des collections : Collection, List, Map, Set, SortedMap, SortedSet

- XXX synchronizedXXX(XXX) pour obtenir une version multi−thread des objets implémentant l'interface XXX
- XXX unmodifiableXXX(XXX) pour obtenir une version non modifiable des objets implémentant l'interface XXX

Exemple (code Java 1.2) :

```
import java.util.*;

public class TestUnmodifiable{
    public static void main(String args[])
    {
      List list = new LinkedList();

      list.add("1");
      list.add("2");
      list = Collections.unmodifiableList(list);

      list.add("3");
    }
}
```

Résultat :

```
C:\>java TestUnmodifiable
Exception in thread "main" java.lang.UnsupportedOperationException
        at java.util.Collections$UnmodifiableCollection.add(Unknown Source)
        at TestUnmodifiable.main(TestUnmodifiable.java:13)
```

L'utilisation d'une méthode synchronizedXXX() renvoie une instance de l'objet qui supporte la synchronisation pour les opérations d'ajout et de suppression d'éléments. Pour le parcours de la collection avec un objet Iterator, il est nécessaire de synchroniser le bloc de code utilisé pour le parcours. Il est important d'inclure aussi dans ce bloc l'appel à la méthode pour obtenir l'objet de type Iterator utilisé pour le parcours.

Exemple (code Java 1.2) :

```
import java.util.*;

public class TestSynchronized{
    public static void main(String args[])
    {
      List maList = new LinkedList();

      maList.add("1");
      maList.add("2");
      maList.add("3");
      maList = Collections.synchronizedList(maList);

      synchronized(maList) {
          Iterator i = maList.iterator();
          while (i.hasNext())
               System.out.println(i.next());
          }
     }
}
```

18.8. Les exceptions du framework

L'exception de type UnsupportedOperationException est levée lorsque qu'une opération optionnelle n'est pas supportée par l'objet qui gère la collection.

L'exception ConcurrentModificationException est levée lors du parcours d'une collection avec un objet Iterator et que cette collection subi une modification structurelle.

19. Les flux

Un programme a souvent besoin d'échanger des informations pour recevoir des données d'une source ou pour envoyer des données vers un destinataire.

La source et la destination de ces échanges peuvent être de nature multiple : un fichier, une socket réseau, un autre programme, etc ...

De la même façon, la nature des données échangées peut être diverse : du texte, des images, du son, etc ...

Ce chapitre contient plusieurs sections :

19.1. Présentation des flux

Les flux (stream en anglais) permettent d'encapsuler ces processus d'envoi et de réception de données. Les flux traitent toujours les données de façon séquentielle.

En java, les flux peuvent être divisés en plusieurs catégories :

- les flux d'entrée (input stream) et les flux de sortie (output stream)
- les flux de traitement de caractères et les flux de traitement d'octets

Java définit des flux pour lire ou écrire des données mais aussi des classes qui permettent de faire des traitements sur les données du flux. Ces classes doivent être associées à un flux de lecture ou d'écriture et sont considérées comme des filtres. Par exemple, il existe des filtres qui permettent de mettre les données traitées dans un tampon (buffer) pour les traiter par lots.

Toutes ces classes sont regroupées dans le package java.io.

19.2. Les classes de gestion des flux

Ce qui déroute dans l'utilisation de ces classes, c'est leur nombre et la difficulté de choisir celle qui convient le mieux en fonction des besoins. Pour faciliter ce choix, il faut comprendre la dénomination des classes : cela permet de sélectionner la ou les classes adaptées aux traitements à réaliser.

Le nom des classes se décompose en un préfixe et un suffixe. Il y a quatre suffixes possibles en fonction du type de flux (flux d'octets ou de caractères) et du sens du flux (entrée ou sortie).

	Flux d'octets	Flux de caractères

Flux d'entrée	InputStream	Reader
Flux de sortie	OutputStream	Writer

Il exite donc quatre hiérarchies de classes qui encapsulent des types de flux particuliers. Ces classes peuvent être séparées en deux séries de deux catégories différentes : les classes de lecture et d'écriture et les classes permettant la lecture de caractères ou d'octets.

- les sous classes de Reader sont des types de flux en lecture sur des ensembles de caractères
- les sous classes de Writer sont des types de flux en écriture sur des ensembles de caractères
- les sous classes de InputStream sont des types de flux en lecture sur des ensembles d'octets
- les sous classes de OutputStream sont des types de flux en écriture sur des ensembles d'octets

Pour le préfixe, il faut distinguer les flux et les filtres. Pour les flux, le préfixe contient la source ou la destination selon le sens du flux.

Préfixe du flux	source ou destination du flux
ByteArray	tableau d'octets en mémoire
CharArray	tableau de caractères en mémoire
File	fichier
Object	objet
Pipe	pipeline entre deux threads
String	chaîne de caractères

Pour les filtres, le préfixe contient le type de traitement qu'il effectue. Les filtres n'existent pas obligatoirement pour des flux en entrée et en sortie.

Type de traitement	Préfixe de la classe	En entrée	En sortie
Mise en tampon	Buffered	Oui	Oui
Concaténation de flux	Sequence	Oui pour flux d'octets	Non
Conversion de données	Data	Oui pour flux d'octets	Oui pour flux d'octets
Numérotation des lignes	LineNumber	Oui pour les flux de caractères	Non
Lecture avec remise dans le flux des données	PushBack	Oui	Non
Impression	Print	Non	Oui
Sérialisation	Object	Oui pour flux d'octets	Oui pour flux d'octets
Conversion octets/caractères	InputStream / OutputStream	Oui pour flux d'octets	Oui pour flux d'octets

- Buffered : ce type de filtre permet de mettre les données du flux dans un tampon. Il peut être utilisé en entrée et en sortie
- Sequence : ce filtre permet de fusionner plusieurs flux.
- Data : ce type de flux permet de traiter les octets sous forme de type de données
- LineNumber : ce filtre permet de numéroter les lignes contenues dans le flux
- PushBack : ce filtre permet de remettre des données lues dans le flux
- Print : ce filtre permet de réaliser des impressions formatées
- Object : ce filtre est utilisé par la sérialisation
- InputStream / OuputStream : ce filtre permet de convertir des octets en caractères

La package java.io définit ainsi plusieurs classes :

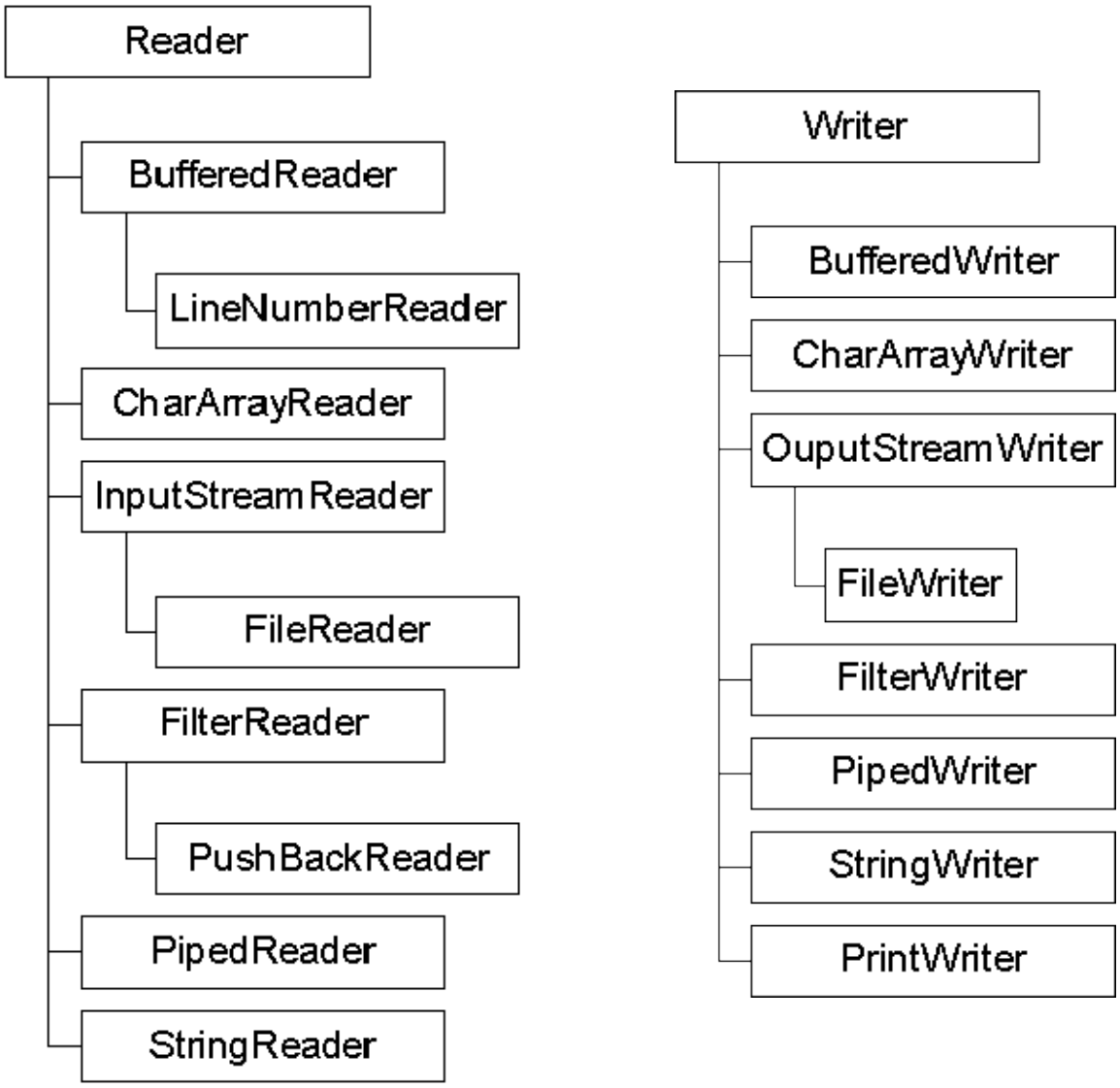
	Flux en lecture	Flux en sortie
Flux de caractères	BufferedReader CharArrayReader FileReader InputStreamReader LineNumberReader PipedReader PushbackReader StringReader	BufferedWriter CharArrayWriter FileWriter OutputStreamWriter PipedWriter StringWriter
Flux d'octets	BufferedInputStream ByteArrayInputStream DataInputStream FileInputStream ObjectInputStream PipedInputStream PushbackInputStream SequenceInputStream	BufferedOutputStream ByteArrayOutputStream DataOuputStream FileOutputStream ObjetOutputStream PipedOutputStream PrintStream

19.3. Les flux de caractères

Ils transportent des données sous forme de caractères : java les gèrent avec le format Unicode qui code les caractères sur 2 octets.

Ce type de flux a été ajouté à partir du JDK 1.1.

Les classes qui gèrent les flux de caractères héritent d'une des deux classes abstraites Reader ou Writer. Il existe de nombreuses sous classes pour traiter les flux de caractères.



19.3.1. La classe Reader

C'est une classe abstaite qui est la classe mère de toutes les classes qui gèrent des flux de caractères en lecture.

Cette classe définit plusieurs méthodes :

Méthodes	Rôles
boolean markSupported()	indique si le flux supporte la possibilité de marquer des positions
boolean ready()	indique si le flux est prêt à être lu
close()	ferme le flux et libère les ressources qui lui étaient associées
int read()	renvoie le caractère lu ou −1 si la fin du flux est atteinte.
int read(char[])	lire plusieurs caractères et les mettre dans un tableau de caractères
int read(char[], int, int)	lire plusieurs caractères. Elle attend en paramètre : un tableau de caractères qui contiendra les caractères lus, l'indice du premier élément du tableau qui recevra le premier caractère et le nombre de caractères à lire. Elle renvoie le nombre de caractères lus ou −1 si aucun caractère n'a été lu. La tableau de caractères contient les caractères lus.

long skip(long)	saute autant de caractères dans le flux que la valeur fournie en paramètre. Elle renvoie le nombre de caractères sautés.
mark()	permet de marquer une position dans le flux
reset()	retourne dans le flux à la dernière position marquée

19.3.2. La classe Writer

C'est une classe abstaite qui est la classe mère de toutes les classes qui gèrent des flux de caractères en écriture.

Cette classe définit plusieurs méthodes :

Méthodes	Rôles
close()	ferme le flux et libère les ressources qui lui étaient associées
write(int)	écrire le caractère en paramètre dans le flux.
write(char[])	écrire le tableau de caractères en paramètre dans le flux.
write(char[], int, int)	écrire plusieurs caractères. Elle attend en paramètres : un tableau de caractères, l'indice du premier caractère dans le tableau à écrire et le nombre de caractères à écrire.
write(String)	écrire la chaîne de caractères en paramètre dans le flux
write(String, int, int)	écrire une portion d'une chaîne de caractères. Elle attend en paramètre : une chaîne de caractères, l'indice du premier caractère dans la chaîne à écrire et le nombre de caractères à écrire.

19.3.3. Les flux de caractères avec un fichier

Les classes FileReader et FileWriter permettent de gérer des flux de caractères avec des fichiers.

19.3.3.1. Les flux de caractères en lecture sur un fichier

Il faut instancier un objet de la classe FileReader. Cette classe hérite de la classe InputStreamReader et possède plusieurs constructeurs qui peuvent tous lever une exception de type FileNotFoundException:

Constructeur	Rôle
FileInputReader(String)	Créer un flux en lecture vers le fichier dont le nom est précisé en paramètre.
FileInputReader(File)	Idem mais le fichier est précisé avec un objet de type File

Exemple (code Java 1.1) :

```
FileReader fichier = new FileReader(«monfichier.txt»);
```

Il existe plusieurs méthodes de la classe FileReader qui permettent de lire un ou plusieurs caractères dans le flux. Toutes ces méthodes sont héritées de la classe Reader et peuvent toutes lever l'exception IOException.

Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

19.3.3.2. Les flux de caractères en écriture sur un fichier

Il faut instancier un objet de la classe FileWriter qui hérite de la classe OuputStreamWriter. Cette classe possède plusieurs constructeurs :

Constructeur	Rôle
FileWriter(String)	Si le nom du fichier précisé n'existe pas alors le fichier sera créé. Si il existe et qu'il contient des données celles ci seront écrasées.
FileWriter(File)	Idem mais le fichier est précisé avec un objet de la classe File.
FileWriter(String, boolean)	Le booléen permet de préciser si les données seront ajoutées au fichier (valeur true) ou écraseront les données existantes (valeur false)

Exemple (code Java 1.1) :

```
FileWriter fichier = new FileWriter («monfichier.dat»);
```

Il existe plusieurs méthodes de la classe FileWriter héritées de la classe Writer qui permettent d'écrire un ou plusieurs caractères dans le flux.

Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

19.3.4. Les flux de caractères tamponnés avec un fichier.

Pour améliorer les performances des flux sur un fichier, la mise en tampon des données lues ou écrites permet de traiter un ensemble de caractères représentant une ligne plutôt que de traiter les données caractères par caractères. Le nombre d'opérations est ainsi réduit.

Les classes BufferedReader et BufferedWriter permettent de gérer des flux de caractères tamponnés avec des fichiers.

19.3.4.1. Les flux de caractères tamponnés en lecture avec un fichier

Il faut instancier un objet de la classe BufferedReader. Cette classe possède plusieurs constructeurs qui peuvent tous lever une exception de type FileNotFoundException:

Constructeur	Rôle
BufferedReader(Reader)	le paramètre fourni doit correspondre au flux à lire.
BufferedReader(Reader, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception de type IllegalArgumentException est levée.

Exemple (code Java 1.1) :

```
BufferedReader fichier  = new BufferedReader(new FileReader("monfichier.txt"));
```

Il existe plusieurs méthodes de la classe BufferedReader héritées de la classe Reader qui permettent de lire un ou plusieurs caractères dans le flux. Toutes ces méthodes peuvent lever une exception de type IOException. Elle définit une méthode supplémentaire pour la lecture :

Méthode	Rôle
String readLine()	lire une ligne de caractères dans le flux. Une ligne est une suite de caractères qui se termine par un retour chariot '\r' ou un saut de ligne '\n' ou les deux.

La classe BufferedReader possède plusieurs méthodes pour gérer le flux hérité de la classe Reader.

Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

Exemple (code Java 1.1) :

```
import java.io.*;

public class TestBufferedReader {
  protected String source;

  public TestBufferedReader(String source) {
    this.source = source;
    lecture();
  }

  public static void main(String args[]) {
    new TestBufferedReader("source.txt");
  }

  private void lecture() {
    try {
      String ligne ;
      BufferedReader fichier = new BufferedReader(new FileReader(source));

      while ((ligne = fichier.readLine()) != null) {
          System.out.println(ligne);
      }

      fichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

19.3.4.2. Les flux de caractères tamponnés en écriture avec un fichier

Il faut instancier un objet de la classe BufferedWriter. Cette classe possède plusieurs constructeurs :

Constructeur	Rôle
BufferedWriter(Writer)	le paramètre fourni doit correspondre au flux dans lequel les données sont écrites.
BufferedWriter(Writer, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception IllegalArgumentException est levée.

Exemple (code Java 1.1) :

```
BufferedWriter fichier = new BufferedWriter( new FileWriter(«monfichier.txt»));
```

Il existe plusieurs méthodes de la classe BufferedWriter héritées de la classe Writer qui permettent de lire un ou plusieurs caractères dans le flux.

La classe BufferedWriter possède plusieurs méthodes pour gérer le flux :

Méthode	Rôle
flush()	vide le tampon en écrivant les données dans le flux.
newLine()	écrire un séparateur de ligne dans le flux

Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

Exemple (code Java 1.1) :

```
import java.io.*;
import java.util.*;

public class TestBufferedWriter {
  protected String destination;

  public TestBufferedWriter(String destination) {
    this.destination = destination;
    traitement();
  }

  public static void main(String args[]) {
    new TestBufferedWriter("print.txt");
  }

  private void traitement() {
    try {
      String ligne ;
      int nombre = 123;
      BufferedWriter fichier = new BufferedWriter(new FileWriter(destination));

      fichier.write("bonjour tout le monde");
      fichier.newLine();
      fichier.write("Nous sommes le "+ new Date());
      fichier.write(", le nombre magique est " + nombre);

      fichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

19.3.4.3. La classe PrintWriter

Cette classe permet d'écrire dans un flux des données formatées.

Cette classe possède plusieurs constructeurs :

Constructeur	Rôle
PrintWriter(Writer)	Le paramètre fourni précise le flux. Le tampon est automatiquement vidé.
PrintWriter(Writer, boolean)	Le booléen permet de préciser si le tampon doit être automatiquement vidé

PrintWriter(OutputStream)	Le paramètre fourni précise le flux. Le tampon est automatiquement vidé.
PrintWriter(OutputStream, boolean)	Le booléen permet de préciser si le tampon doit être automatiquement vidé

Exemple (code Java 1.1) :

```
PrintWriter fichier = new PrintWriter( new FileWriter(«monfichier.txt»));
```

Il existe de nombreuses méthodes de la classe PrintWriter qui permettent d'écrire un ou plusieurs caractères dans le flux en les formatant. Les méthodes write() sont héritées de la classe Writer. Elle définit plusieurs méthodes pour envoyer des données formatées dans le flux :

- print(...)

  Plusieurs méthodes print acceptent des données de différents types pour les convertir en caractères et les écrire dans le flux

- println()

  Cette méthode permet de terminer la ligne courante dans le flux en y écrivant un saut de ligne.

- println (...)

  Plusieurs méthodes println acceptent des données de différents types pour les convertir en caractères et les écrire dans le flux avec une fin de ligne.

La classe PrintWriter possède plusieurs méthodes pour gérer le flux :

Méthode	Rôle
flush()	Vide le tampon en écrivant les données dans le flux.

Exemple (code Java 1.1) :

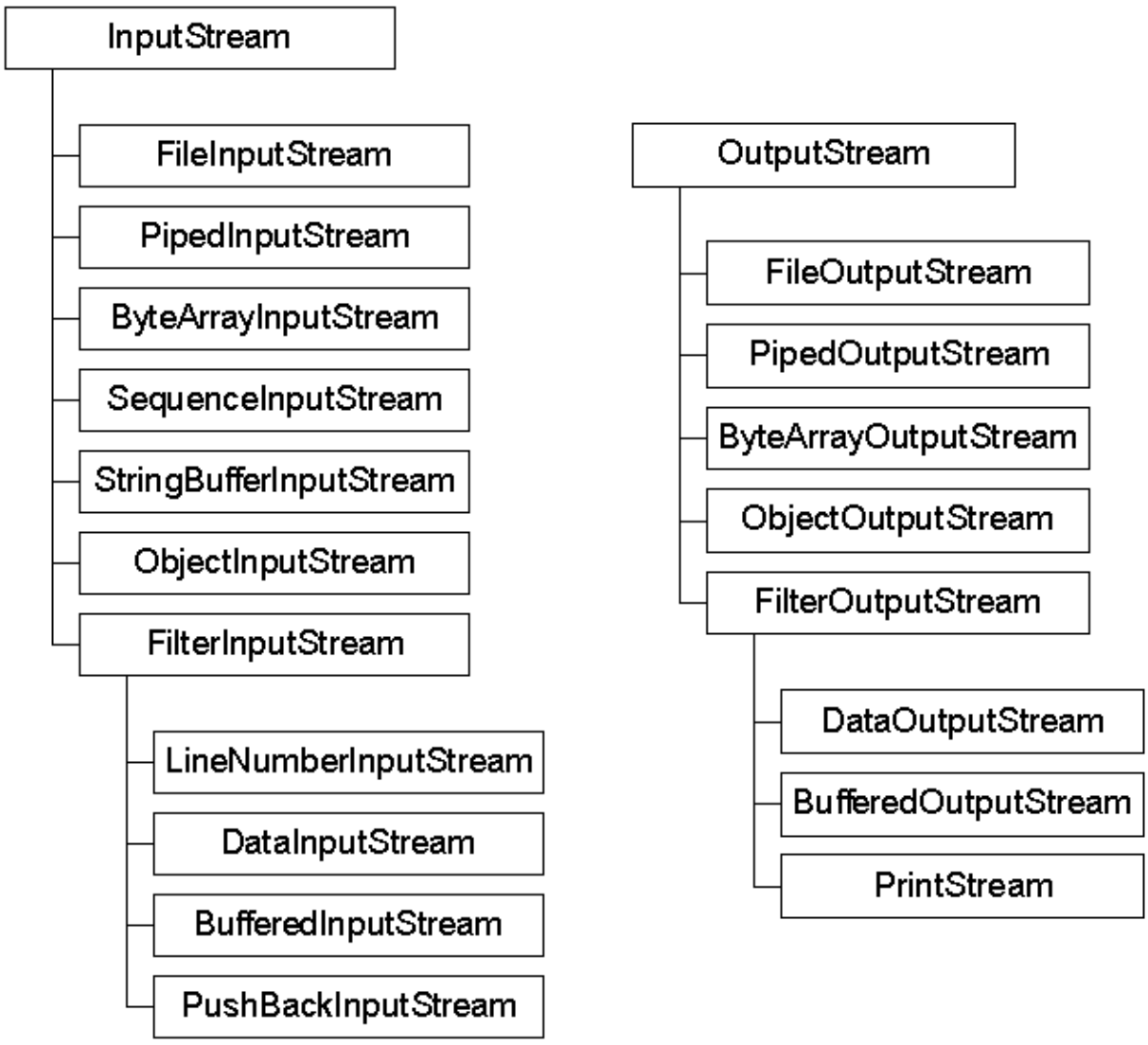
```
import java.io.*;
import java.util.*;

public class TestPrintWriter {
  protected String destination;

  public TestPrintWriter(String destination) {
    this.destination = destination;
    traitement();
  }

  public static void main(String args[]) {
    new TestPrintWriter("print.txt");
  }

  private void traitement() {
    try {
      String ligne ;
      int nombre = 123;
      PrintWriter fichier = new PrintWriter(new FileWriter(destination));

      fichier.println("bonjour tout le monde");
      fichier.println("Nous sommes le "+ new Date());
      fichier.println("le nombre magique est " + nombre);
```

```
      fichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

19.4. Les flux d'octets

Ils transportent des données sous forme d'octets. Les flux de ce type sont capables de traiter toutes les données.

Les classes qui gèrent les flux d'octets héritent d'une des deux classes abstraites InputStream ou OutputStream. Il existe de nombreuses sous classes pour traiter les flux d'octets.



19.4.1. Les flux d'octets avec un fichier.

Les classes FileInputStream et FileOutputStream permettent de gérer des flux d'octets avec des fichiers.

19.4.1.1. Les flux d'octets en lecture sur un fichier

Il faut instancier un objet de la classe FileInputStream. Cette classe possède plusieurs constructeurs qui peuvent tous lever l'exception FileNotFoundException:

Constructeur	Rôle
FileInputStream(String)	Ouvre un flux en lecture sur le fichier dont le nom est donné en paramètre
FileInputStream(File)	Idem mais le fichier est précisé avec un objet de type File

Exemple (code Java 1.1) :

```
FileInputStream fichier = new FileInputStream(«monfichier.dat»);
```

Il existe plusieurs méthodes de la classe FileInputStream qui permettent de lire un ou plusieurs octets dans le flux. Toutes ces méthodes peuvent lever l'exception IOException.

- int read()

  Cette méthode envoie la valeur de l'octet lu ou −1 si la fin du flux est atteinte.

Exemple (code Java 1.1) :

```
int octet = 0;
while (octet != 1 ) {
   octet = fichier.read();
}
```

- int read(byte[], int, int)

  Cette méthode lit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contiendra les octets lus, l'indice du premier élément du tableau qui recevra le premier octet et le nombre d'octets à lire.

  Elle renvoie le nombre d'octets lus ou −1 si aucun octet n'a été lus. La tableau d'octets contient les octets lus.

La classe FileInputStream possède plusieurs méthodes pour gérer le flux :

Méthode	Rôle
long skip(long)	saute autant d'octets dans le flux que la valeur fournie en paramètre. Elle renvoie le nombre d'octets sautés.
close()	ferme le flux et libère les ressources qui lui étaient associées
int available()	retourne le nombre d'octets qu'il est encore possible de lire dans le flux

Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

19.4.1.2. Les flux d'octets en écriture sur un fichier

Il faut instancier un objet de la classe FileOutputStream. Cette classe possède plusieurs constructeurs :

Constructeur	Rôle
FileOutputStream(String)	Si le fichier précisé n'existe pas, il sera créé. Si il existe et qu'il contient des données celles ci seront écrasées.

FileOutputStream(String, boolean)	Le booléen permet de préciser si les données seront ajoutées au fichier (valeur true) ou écraseront les données existantes (valeur false)

Exemple (code Java 1.1) :

```
FileOuputStream fichier = new FileOutputStream(«monfichier.dat»);
```

Il existe plusieurs méthodes de la classe FileOutputStream qui permettent de lire un ou plusieurs octets dans le flux.

- write(int)

  Cette méthode écrit l'octet en paramètre dans le flux.
- write(byte[])

  Cette méthode écrit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contient les octets à écrire : tous les éléments du tableau sont écrits.
- write(byte[], int, int)

  Cette méthode écrit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contient les octets à écrire, l'indice du premier éléments du tableau d'octets à écrire et le nombre d'octets à écrire.

Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

Exemple (code Java 1.1) :

```
import java.io.*;

public class CopieFichier {
  protected String source;
  protected String destination;

  public CopieFichier(String source, String destination) {
    this.source = source;
    this.destination = destination;
    copie();
  }

  public static void main(String args[]) {
    new CopieFichier("source.txt","copie.txt");
  }

  private void copie() {
    try {
      FileInputStream fis = new FileInputStream(source);
      FileOutputStream fos = new FileOutputStream(destination);
      while(fis.available() > 0) fos.write(fis.read());
      fis.close();
      fos.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

19.4.2. Les flux d'octets tamponnés avec un fichier.

Pour améliorer les performances des flux sur un fichier, la mise en tampon des données lues ou écrites permet de traiter un ensemble d'octets plutôt que de traiter les données octets par octets. Le nombre d'opérations est ainsi réduit.

19.5. La classe File

Les fichiers et les répertoires sont encapsulés dans la classe File du package java.io. Il n'existe pas de classe pour traiter les répertoires car ils sont considérés comme des fichiers. Une instance de la classe File est une représentation logique d'un fichier ou d'un répertoire qui peut ne pas exister physiquement sur le disque.

Si le fichier ou le répertoire existe, de nombreuses méthodes de la classe File permettent d'obtenir des informations sur le fichier. Sinon plusieurs méthodes permettent de créer des fichiers ou des répertoires. Voici une liste des principales méthodes :

Méthode	Rôle
boolean canRead()	indique si le fichier peut être lu
boolean canWrite()	indique si le fichier peut être modifié
boolean createNewFile()	 création d'un nouveau fichier vide
File createTempFile(String, String)	 création d'un nouveau fichier dans le répertoire par défaut des fichiers temporaires. Les deux arguments sont le préfixe et le suffixe du fichier.
File createTempFile(String, String, File)	création d'un nouveau fichier temporaire. Les trois arguments sont le préfixe et le suffixe du fichier et le répertoire.
boolean delete()	détruire le fichier ou le repertoire. Le booléen indique le succés de l'opération
deleteOnExit()	 demande la suppression du fichier à l'arrêt de la JVM
boolean exists()	indique si le fichier existe physiquement
String getAbsolutePath()	renvoie le chemin absolu du fichier
String getPath	renvoie le chemin du fichier
boolean isAbsolute()	indique si le chemin est absolu
boolean isDirectory()	indique si le fichier est un répertoire
boolean isFile()	indique si l'objet représente un fichier
long length()	renvoie la longueur du fichier
String[] list()	renvoie la liste des fichiers et répertoire contenu dans le répertoire
boolean mkdir()	création du répertoire
boolean mkdirs()	création du répertoire avec création des répertoires manquants dans l'arborescence du chemin
boolean renameTo()	renommer le fichier

Depuis la version 1.2 du J.D.K., de nombreuses fonctionnalités ont été ajoutées à cette classe :

- la création de fichiers temporaires (createNewFile, createTempFile, deleteOnExit)
- la gestion des attributs "caché" et "lecture seule" (isHidden, isReadOnly)

- des méthodes qui renvoient des objets de type File au lieu de type String (getParentFile, getAbsoluteFile, getCanonicalFile, listFiles)
- une méthode qui renvoie le fichier sous forme d'URL (toURL)

Exemple (code Java 1.1) :

```
import java.io.*;

public class TestFile {
  protected String nomFichier;
  protected File fichier;

  public TestFile(String nomFichier) {
    this.nomFichier = nomFichier;
    fichier = new File(nomFichier);
    traitement();
  }

  public static void main(String args[]) {
    new TestFile(args[0]);
  }

  private void traitement() {

    if (!fichier.exists()) {
      System.out.println("le fichier "+nomFichier+"n'existe pas");
      System.exit(1);
    }

    System.out.println(" Nom du fichier    : "+fichier.getName());
    System.out.println(" Chemin du fichier : "+fichier.getPath());
    System.out.println(" Chemin absolu     : "+fichier.getAbsolutePath());
    System.out.println(" Droit de lecture  : "+fichier.canRead());
    System.out.println(" Droite d'ecriture : "+fichier.canWrite());

    if (fichier.isDirectory() ) {
      System.out.println(" contenu du repertoire ");
      String fichiers[] = fichier.list();
      for(int i = 0; i <fichiers.length; i++) System.out.println("  "+fichiers[i]);
    }
  }
}
```

Exemple (code Java 1.2) :

```
import java.io.*;

public class TestFile_12 {
  protected String nomFichier;
  protected File fichier;

  public TestFile_12(String nomFichier) {
    this.nomFichier = nomFichier;
    fichier = new File(nomFichier);
    traitement();
  }

  public static void main(String args[]) {
    new TestFile_12(args[0]);
  }

  private void traitement() {

    if (!fichier.exists()) {
      System.out.println("le fichier "+nomFichier+"n'existe pas");
      System.exit(1);
    }

    System.out.println(" Nom du fichier    : "+fichier.getName());
    System.out.println(" Chemin du fichier : "+fichier.getPath());
    System.out.println(" Chemin absolu     : "+fichier.getAbsolutePath());
    System.out.println(" Droit de lecture  : "+fichier.canRead());
```

```
    System.out.println(" Droite d'ecriture : "+fichier.canWrite());

    if (fichier.isDirectory() ) {
      System.out.println(" contenu du repertoire ");
      File fichiers[] = fichier.listFiles();
      for(int i = 0; i <fichiers.length; i++) {

        if (fichiers[i].isDirectory())
          System.out.println("  ["+fichiers[i].getName()+"]");
        else
          System.out.println("  "+fichiers[i].getName());
      }
    }
  }
}
```

19.6. Les fichiers à accès direct

Les fichiers à accès direct permettent un accès rapide à un enregistrement contenu dans un fichier. Le plus simple pour utiliser un tel type de fichier est qu'il contienne des enregistrements de taille fixe mais ce n'est pas obligatoire. Il est possible dans un tel type de fichier de mettre à jour directement un de ces enregistrements.

La classe RamdonAccessFile encapsule les opérations de lecture/écriture d'un tel fichier. Elle implémente les interfaces DataInput et DataOutput.

Elle possède deux contructeurs qui attendent en paramètre le fichier à utiliser (sous la forme d'un nom de fichier ou d'un objet de type File qui encapsule le fichier) et le mode d'accès.

Le mode est une chaîne de caractères qui doit être égal à «r» ou «rw» selon que le mode soit lecture seule ou lecture/écriture.

Ces deux contructeurs peuvent lever les exceptions suivantes :

- FileNotFoundException si le fichier n'est pas trouvé
- IllegalArgumentException si le mode n'est pas «r» ou «rw»
- SecurityException si le gestionnaire de sécurité empêche l'accès aux fichiers dans le mode précisé

La classe RandomAccessFile possède de nombreuses méthodes writeXXX() pour écrire des types primitifs dans le fichier.

Exemple :

```
package com.moi.test;

import java.io.RandomAccessFile;

public class TestRandomAccesFile {
  public static void main(String[] args) {
    try {
      RandomAccessFile monFichier = new RandomAccessFile("monfichier.dat", "rw");
      for (int i = 0; i < 10; i++) {
        monFichier.writeInt(i * 100);
      }
      monFichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Elle possède aussi de nombreuses classes readXXX() pour lire des données primitives dans le fichier.

Exemple :

```
package com.moi.test;

import java.io.RandomAccessFile;

public class TestRandomAccesFile {
  public static void main(String[] args) {
    try {
      RandomAccessFile monFichier = new RandomAccessFile("monfichier.dat", "rw");
      for (int i = 0; i < 10; i++) {
         System.out.println(monFichier.readInt());
      }
      monFichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Résultat :

```
 0
  100
  200
  300
  400
  500
  600
  700
  800
  900
```

Pour naviguer dans le fichier, la classe utilise un pointeur qui indique la position dans le fichier ou les opérations de lecture ou de mise à jour doivent être effectuées. La méthode getFilePointer() permet de connaître la position de ce pointeur et la méthode seek() permet de le déplacer.

La méthode seek() attendant en paramètre un entier long qui réprésentent la position, dans le fichier, précisée en octets. La première position commence à zéro.

Exemple : lecture de la sixème données

```
package com.moi.test;

import java.io.RandomAccessFile;

public class TestRandomAccesFile {
  public static void main(String[] args) {
    try {
      RandomAccessFile monFichier = new RandomAccessFile("monfichier.dat", "rw");
      // 5 représente le sixième enregistement puisque le premier commence à 0
      // 4 est la taille des données puisqu'elles sont des entiers de type int
      // (codé sur 4 octets)
      monFichier.seek(5*4);
      System.out.println(monFichier.readInt());
      monFichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Résutlat :

```
500
```

20. La sérialisation

La sérialisation est un procédé introduit dans le JDK version 1.1 qui permet de rendre un objet persistant. Cet objet est mis sous une forme sous laquelle il pourra être reconstitué à l'identique. Ainsi il pourra être stocké sur un disque dur ou transmis au travers d'un réseau pour le créer dans une autre JVM. C'est le procédé qui est utilisé par RMI. La sérialisation est aussi utilisée par les beans pour sauvegarder leurs états.

Au travers de ce mécanisme, Java fourni une façon facile, transparente et standard de réaliser cette opération : ceci permet de facilement mettre en place un mécanisme de persistance. Il est de ce fait inutile de créer un format particulier pour sauvegarder et relire un objet. Le format utilisé est indépendant du système d'exploitation. Ainsi, un objet sérialisé sur un système peut être réutilisé par un autre système pour récréer l'objet.

L'ajout d'un attribut à l'objet est automatiquement pris en compte lors de la sérialisation. Attention toutefois, la déserialisation de l'objet doit se faire avec la classe qui a été utilisée pour la sérialisation.

La sérialisation peut s'appliquer facilement à tous les objets.

Ce chapitre contient plusieurs sections :

20.1. Les classes et les interfaces de la sérialisation

La sérialisation utilise l'interface Serializable et les classes ObjectOutputStream et ObjectInputStream

20.1.1. L'interface Serializable

Cette interface ne définit aucune méthode mais permet simplement de marquer une classe comme pouvant être sérialisée.

Tout objet qui doit être sérialisé doit implémenter cette interface ou une de ses classes mères doit l'implémenter.

Si l'on tente de sérialiser un objet qui n'implémente pas l'interface Serializable, une exception NotSerializableException est levée.

Exemple (code Java 1.1) : une classe serializable possédant trois attributs

```
public class Personne implements java.io.Serializable {
  private String nom = "";
  private String prenom = "";
  private int taille = 0;

  public Personne(String nom, String prenom, int taille) {
    this.nom = nom;
    this.taille = taille;
    this.prenom = prenom;
```

```
  }

  public String getNom() {
    return nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
  }

  public int getTaille() {
    return taille;
  }

  public void setTaille(int taille) {
    this.taille = taille;
  }

  public String getPrenom() {
    return prenom;
  }

  public void setPrenom(String prenom) {
    this.prenom = prenom;
  }
}
```

20.1.2. La classe ObjectOuputStream

Cette classe permet de sérialiser un objet.

Exemple (code Java 1.1) : sérialisation d'un objet et enregistrement sur le disque dur

```
import java.io.*;

public class SerializerPersonne {

  public static void main(String argv[]) {
    Personne personne = new Personne("Dupond","Jean",175);
    try {
      FileOutputStream fichier = new FileOutputStream("personne.ser");
      ObjectOutputStream oos = new ObjectOutputStream(fichier);
      oos.writeObject(personne);
      oos.flush();
      oos.close();
    }
    catch (java.io.IOException e) {
      e.printStackTrace();
    }
  }
}
```

On définit un fichier avec la classe FileOutputStream. On instancie un objet de classe ObjectOutputStream en lui fournissant en paramètre le fichier : ainsi, le résultat de la sérialisation sera envoyé dans le fichier.

On appelle la méthode writeObject en lui passant en paramètre l'objet à sérialiser. On appelle la méthode flush() pour vider le tampon dans le fichier et la méthode close() pour terminer l'opération.

Lors de ces opérations une exception de type IOException peut être levée si un problème intervient avec le fichier.

Après l'exécution de cet exemple, un fichier nommé « personne.ser » est créé. On peut visualiser son contenu mais surtout pas le modifier car sinon il serait corrompu. En effet, les données contenues dans ce fichier ne sont pas toutes au format caractères.

La classe ObjectOutputStream contient aussi plusieurs méthodes qui permettent de sérialiser des types élémentaires et

non des objets : writeInt, writeDouble, writeFloat ...

Il est possible dans un même flux d'écrire plusieurs objets les uns à la suite des autres. Ainsi plusieurs objets peuvent être sauvegardés. Dans ce cas, il faut faire attention de relire les objets dans leur ordre d'écriture.

20.1.3. La classe ObjectInputStream

Cette classe permet de déssérialiser un objet.

Exemple (code Java 1.1) :

```
import java.io.*;

public class DeSerializerPersonne {

  public static void main(String argv[]) {
    try {
      FileInputStream fichier = new FileInputStream("personne.ser");
      ObjectInputStream ois = new ObjectInputStream(fichier);
      Personne personne = (Personne) ois.readObject();
      System.out.println("Personne : ");
      System.out.println("nom : "+personne.getNom());
      System.out.println("prenom : "+personne.getPrenom());
      System.out.println("taille : "+personne.getTaille());
    }
    catch (java.io.IOException e) {
      e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
   }
}
```

Résultat :

```
C:\dej>java DeSerializerPersonne
Personne :
nom : Dupond
prenom : Jean
taille : 175
```

On créer un objet de la classe FileInputStream qui représente le fichier contenant l'objet sérialisé. On créer un objet de type ObjectInputStream en lui passant le fichier en paramètre. Un appel à la méthode readObject() retourne l'objet avec un type Object. Un cast est nécessaire pour obtenir le type de l'objet. La méthode close() permet de terminer l'opération.

Si la classe a changée entre le moment ou elle a été sérialisée et le moment ou elle est déserialisée, une exception est levée :

Exemple : la classe Personne est modifiée et recompilée

```
C:\temp>java DeSerializerPersonne
java.io.InvalidClassException: Personne; Local class not compatible: stream class
desc serialVersionUID=-2739669178469387642 local class serialVersionUID=39870587
36962107851

atjava.io.ObjectStreamClass.validateLocalClass(ObjectStreamClass.java:4
38)
at java.io.ObjectStreamClass.setClass(ObjectStreamClass.java:482)
at java.io.ObjectInputStream.inputClassDescriptor(ObjectInputStream.java
:785)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:353)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:232)
at java.io.ObjectInputStream.inputObject(ObjectInputStream.java:978)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:369)
```

```
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:232)
at DeSerializerPersonne.main(DeSerializerPersonne.java:9)
```

Une exception de type StreamCorruptedException peut être levée si le fichier a été corrompu par exemple en le modifiant avec un éditeur.

Exemple : les 2 premiers octets du fichier personne.ser ont été modifiés avec un éditeur hexa

```
C:\temp>java DeSerializerPersonne

java.io.StreamCorruptedException: InputStream does not containa serialized object
at java.io.ObjectInputStream.readStreamHeader(ObjectInputStream.java:731)
at java.io.ObjectInputStream.<init>(ObjectInputStream.java:165)
at DeSerializerPersonne.main(DeSerializerPersonne.java:8)
```

Une exception de type ClassNotFoundException peut être levée si l'objet est transtypé vers une classe qui n'existe plus ou pas au moment de l'exécution.

Exemple (code Java 1.1) :

```
C:\temp>rename Personne.class Personne2.class
C:\temp>java DeSerializerPersonne

java.lang.ClassNotFoundException: Personne
at java.io.ObjectInputStream.inputObject(ObjectInputStream.java:981)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:369)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:232)
at DeSerializerPersonne.main(DeSerializerPersonne.java:9)
```

La classe ObjectInputStream possède de la même façon que la classe ObjectOutputStream des méthodes pour lire des données de type primitives : readInt(), readDouble(), readFloat ...

Lors de la deserialisation, le constructeur de l'objet n'est jamais utilisé.

20.2. Le mot clé transient

Le contenu des attributs est visible dans le flux dans lequel est sérialisé l'objet. Il est ainsi possible pour toute personne ayant accès au flux de voir le contenu de chaque attribut même si ceux si sont private. Ceci peut poser des problèmes de sécurité surtout si les données sont sensibles.

Java introduit le mot clé transient qui précise que l'attribut qu'il qualifie ne doit pas être inclus dans un processus de serailisation et donc de deserialisation.

Exemple (code Java 1.1) :

```
...
private transient String codeSecret;
...
```

Lors de la deserialisation, les champs transient sont initialisés avec la valeur null. Ceci peut poser des problèmes à l'objet qui doit gérer cette état pour éviter d'avoir des exceptions de type NullPointerException.

20.3. La sérialisation personnalisée

Il est possible de personnaliser la serialisation d'un objet. Dans ce cas, la classe doit implémenter l'interface Externalizable qui hérite de l'interface Serializable.

20.3.1. L'interface Externalizable

Cette interface définit deux méthode : readExternal() et writeExternal().

Par défaut, la serialisation d'un objet qui implémente cette interface ne prend en compte aucun attribut de l'objet.

Remarque : le mot clé transient est donc inutile avec un classe qui implémente l'interface Externalisable

La suite de cette section sera développée dans une version future de ce document

21. L'interaction avec le réseau

Ce chapitre contient plusieurs sections :

21.1. Introduction

Depuis son origine, Java fournit plusieurs classes et interfaces destinées à faciliter l'utilisation du réseau par programmation.

Le modèle OSI (Open System Interconnection) propose un découpage en sept couches des différents composants qui permettent la communication sur un réseau.

Couche	Représentation physique ou logicielle
Application	Netscape ou Internet Explorer ou une application
Présentation	Window, Mac OS ou Unix
Session	WinSock, MacTCP
Transport	TCP / UDP
Réseau	IP
Liaison	PPP, ethernet
Physique	Les cables et les cartes électroniques

Le protocole IP est un protocole de niveau réseau qui permet d'échanger des paquets d'octets appelés datagrammes. Ce protocole ne garantit pas l'arrivée à bon port des messages. Cette fonctionnalité peut être implémentée par la couche supérieure, comme par exemple avec TCP. Un datagramme IP est l'unité de transfert à ce niveau. Cette série d'octets contient les informations du message, un en tête (adresse source de destination, ...). mais aussi des informations de contrôle. Ces informations permettent aux routeurs de faire transiter les paquets sur l'internet.

La couche de transport est implémentée dans les protocoles UDP ou TCP. Ils permettent la communication entre des applications sur des machines distantes.

La notion de service permet à une même machine d'assurer plusieurs communications simultanément.

Le système des sockets est le moyen de communication inter−processus développé pour l'Unix Berkeley (BSD). Il est actuellement implémenté sur tous les systèmes d'exploitation utilisant TCP/IP. Une socket est le point de communication par lequel un thread peut émettre ou recevoir des informations et ainsi elle permet la communication entre deux applications à travers le réseau.

La communication se fait sur un port particulier de la machine. Le port est une entité logique qui permet d'associer un service particulier à une connection. Un port est identifié par un entier de 1 à 65535. Par convention les 1024 premiers sont réservés pour des services standard (80 : HTTP, 21 : FTP, 25: SMTP, ...)

Java prend en charge deux protocoles : TCP et UDP.

Les classes et interfaces utiles au développement réseau sont regroupés dans le package java.net.

21.2. Les adresses internet

Une adresse internet permer d'identifier de façon unique une machine sur un réseau. Cette adresse pour le protocole I.P. est sous la forme de quatre octets séparés chacun par un point. Chacun de ces octets appartient à une classe selon l'étendue du réseau.

Pour faciliter la compréhension humaine, un serveur particulier appelé DNS (Domaine Name Service) est capable d'associer un nom à une adresse I.P.

21.2.1. La classe InetAddress

Une adresse internet est composée de quatre octets séparés chacun par un point.

Un objet de la classe InetAddress représente une adresse Internet. Elle contient des méthodes pour lire une adresse, la comparer avec une autre ou la convertir en chaîne de caractères. Elle ne possède pas de constructeur : il faut utiliser certaines méthodes statiques de la classe pour obtenir une instance de cette classe.

La classe InetAdress encapsule des fonctionnalités pour utiliser les adresses internet. Elle ne possède pas de constructeur mais propose trois méthodes statiques :

Méthode	Rôle
InetAddress getByName(String)	Renvoie l'adresse internet associée au nom d'hote fourni en paramètre
InetAddress[] getAllByName(String)	Renvoie un tableau des adresses internet associées au nom d'hote fourni en paramètre
InetAddress getLocalHost()	Renvoie l'adresse internet de la machine locale
byte[] getAddress()	Renvoie un tableau contenant les 4 octets de l'adresse internet
String getHostAddress()	Renvoie l'adresse internet sous la forme d'une chaîne de caractères
String getHostName()	Renvoie le nom du serveur

Exemple :

```
import java.net.*;

public class TestNet1 {

  public static void main(String[] args) {
        try {
      InetAddress adrLocale = InetAddress.getLocalHost();
      System.out.println("Adresse locale = "+adrLocale.getHostAddress());
      InetAddress adrServeur = InetAddress.getByName("java.sun.com");
      System.out.println("Adresse Sun = "+adrServeur.getHostAddress());
      InetAddress[] adrServeurs = InetAddress.getAllByName("www.microsoft.com");
      System.out.println("Adresses Microsoft : ");
      for (int i = 0; i <adrServeurs.length; i++) {
        System.out.println("     "+adrServeurs[i].getHostAddress());
      }
        } catch (Exception e) {
          e.printStackTrace();
```

```
        }

  }
}
```

Résultat :

```
Adresse locale = 192.166.23.103
Adresse Sun = 192.18.97.71
Adresses Microsoft :
     207.46.249.27
     207.46.134.155
     207.46.249.190
     207.46.134.222
     207.46.134.190
```

21.3. L'accès aux ressources avec une URL

Une URL (Uniform Resource Locator) ou locateur de ressource uniforme est une chaîne de caractères qui désigne une ressource précise accessible par Internet ou Intranet. Une URL est donc une référence à un objet dont le format dépend du protocol utilisé pour accéder à la ressource.

Dans le cas du protocol http, l'URL est de la forme :

http://<serveur>:<port>/<chemin>?<param1>&<param2>

Elle se compose du protocole (HTTP), d'une adresse IP ou du nom de domaine du serveur de destination, avec éventuellement un port, un chemin d'accès vers un fichier ou un nom de service et éventuellement des paramètres sous la forme clé=valeur.

Dans le cas du protocol ftp, l'URL est de la forme :

ftp://<user>:<motdepasse>@<serveur>:<port>/<chemin>

Dans le cas d'un e−mail, l'URL est de la forme

mailto:<email>

Dans le cas d'un fichier local, l'URL est de la forme :

file://<serveur>/<chemin>

Elle se compose de la désignation du serveur (non utilisé dans le cas du système de fichier local) et du chemin absolu de la ressource.

21.3.1. La classe URL

Un objet de cette classe encapsule une URL : la validité syntaxique de l'URL est assurée mais l'existence de la ressource représentée part l'URL ne l'est pas.

Exemple d'URL :

http: //www.test.fr :80 /images/image.gif

Dans l'exemple, http represente le protocole, www.test.fr représente le serveur, 80 représente le port, /images/image.gif représente la ressource.

Le nom du protocole indique au navigateur le protocole qui doit être utilisé pour accéder à la ressource. Il existe plusieurs protocoles sur internet : http, ftp, smtp ...

L'identification du serveur est l'information qui désigne une machine sur le réseau, identifiée par une adresse IP. Cette adresse s'écrit sous la forme de quatre entiers séparés par un point. Une machine peut se voir affecter un nom logique (hostname) composé d'un nom de machine (ex : www), d'un nom de sous domaine (ex : toto) et d'un nom de domaine (ex :fr). Chaque domaine possède un serveur de nom (DNS : Domain Name Server) chargé d'effectuer la correspondanre entre les noms logiques et les adresses IP.

Le numéro de port désigne le service. En mode client/serveur, un client s'adresse à un programme particulier (le service) qui s'exécute sur le serveur. Le numéro de port identifie ce service. Cette information est facultative dans l'URL : par exemple si aucun numéro n'est précisé dans une url, un browser dirige sa demande vers un port standard : par défaut, le service http est associé au port 80, le service ftp au port 21, etc ...

L'identification de la ressource indique le chemin d'accès de celle ci sur le serveur.

Le constructeur de la classe lève une exception du type MalformedURLException si la syntaxe de l'URL n'est pas correcte.

Les objets créés sont constants et ne peuvent plus être modifiés par la suite.

Exemple :

```
URL pageURL = null;
 try {
   pageURL = new URL(getDocumentBase( ), "http://www.javasoft.com");
 } catch (MalformedURLException mue) {}
```

La classe URL possède des getters pour obtenir les différents éléments qui composent l'URL : getProtocole(), getHost(), getPort(), getFile().

La méthode openStream() ouvre un flux de données en entrée pour lire la ressource et renvoie un objet de type InputStream.

La méthode openConnection ouvre une connexion vers la ressource et renvoie un objet de type URLConnexion

21.3.2. La classe URLConnection

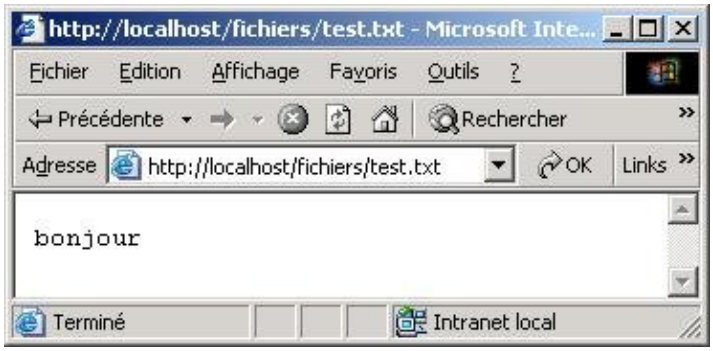
Cette classe abstraite encapsule une connexion vers une ressource désignée par une URL pour obtenir un flux de données ou des informations sur la ressource.

Exemple :

```
import java.net.*;
import java.io.*;

public class TestURLConnection {

        public static void main(String[] args) {

                String donnees;

                try {

                        URL monURL = new URL("http://localhost/fichiers/test.txt");

                        URLConnection connexion = monURL.openConnection();
                        InputStream flux = connexion.getInputStream();

                        int donneesALire = connexion.getContentLength();

                        for(;donneesALire != 0; donneesALire--)
```

```
                        System.out.print((char)flux.read());

                        // Fermeture de la connexion
                        flux.close();

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

}
```

Pour cet exemple, le fichier test.txt doit être accessible via le serveur web dans le répertoire "fichiers".



21.3.3. La classe URLEncoder

Cette classe est une classe utilitaire qui propose la méthode statique encode() pour encoder une URL. Elle remplace notamment les espaces par un signe "+" et les caractères spéciaux par un signe "%" suivi du code du caractère.

Exemple :

```
import java.net.*;

public class TestEncodeURL {

  public static void main(String[] args) {
        String url = "http://www.test.fr/images perso/mon image.gif";
        System.out.println(URLEncoder.encode(url));
  }
}
```

Résultat :

```
http%3A%2F%2Fwww.test.fr%2Fimages+perso%2Fmon+image.gif
```

Depuis le JDK 1.4, il existe une version surchargée de la méthode encode() qui nécessite le passage d'un paramètre supplémentaire : une chaîne de caractère qui précise le format d'encodage des caractères. Cette méthode remplace l'ancienne méthode encode() qui est dépréciée. Elle peut lever une exception du type UnsupportedEncodingException.

Exemple (JDK 1.4) :

```
import java.io.UnsupportedEncodingException;
import java.net.*;

public class TestEncodeURL {

  public static void main(String[] args) {
        try {
        String url = "http://www.test.fr/images perso/mon image.gif";
        System.out.println(URLEncoder.encode(url,"UTF-8"));
```

```
        System.out.println(URLEncoder.encode(url,"UTF-16"));
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        }
  }
}
```

Résultat :

```
http%3A%2F%2Fwww.test.fr%2Fimages+perso%2Fmon+image.gif
http%FE%FF%00%3A%00%2F%00%2Fwww.test.fr%FE%FF%00%2Fimages+perso%FE%FF%00%2Fmon+image.gif
```

21.3.4. La classe HttpURLConnection

Cette classe qui hérite de URLConnection encapsule une connexion utilisant le protocole HTTP.

Exemple :

```
import java.net.*;
import java.io.*;

public class TestHttpURLConnection {

  public static void main(String[] args) {
    HttpURLConnection connexion = null;

   try {
    URL url = new URL("http://java.sun.com");

    System.out.println("Connexion a l'url ...");
    connexion = (HttpURLConnection) url.openConnection();

    connexion.setAllowUserInteraction(true);
    DataInputStream in = new DataInputStream(connexion.getInputStream());

      if (connexion.getResponseCode() != HttpURLConnection.HTTP_OK) {
        System.out.println(connexion.getResponseMessage());
      } else {
        while (true) {
          System.out.print((char) in.readUnsignedByte());
        }
      }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
      connexion.disconnect();
      }
    System.exit(0);
  }
}
```

21.4. Utilisation du protocole TCP

TCP est un protocole qui permet une connection de type point à point entre deux applications. C'est un protocole fiable qui garantit la réception dans l'ordre d'envoi des données. En contre partie, ce protocole offre de moins bonnes performances mais c'est le prix à payer pour la fiabilité.

TCP utilise la notion de port pour permettre à plusieurs applications d'utiliser TCP.

Dans une liaison entre deux ordinateurs, l'un des deux joue le rôle de serveur et l'autre celui de client.

21.4.1. La classe SocketServer

La classe ServerSocket est utilisée côté serveur : elle attend simplement les appels du ou des clients. C'est un objet du type Socket qui prend en charge la transmission des données.

Cette classe représente la partie serveur du socket. Un objet de cette classe est associé à un port sur lequel il va attendre les connexions d'un client. Généralement, à l'arrivée d'une demande de connexion, un thread est lancé pour assurer le dialogue avec le client sans bloquer les connexions des autres clients.

La classe SocketServer possède plusieurs constructeurs dont les principaux sont :

Constructeur	Rôle
ServerSocket()	Constructeur par défaut
ServerSocket(int)	Créer une socket sur le port fourni en paramètre
ServerSocket(int, int)	Créer une socket sur le port avec la taille maximale de la file fourni en paramètre

Tous ces constructeurs peuvent lever une exception de type IOException.

La classe SocketServer possède plusieurs méthodes :

Méthode	Rôle
Socket accept()	Attendre une nouvelle connexion
void close()	Fermer la socket

Si un client tente de communiquer avec le serveur, la méthode accept() renvoie une socket qui encapsule la communication avec ce client.

Le mise en oeuvre de la classe SocketServer suit toujours la même logique :

- créer une instance de la classe SocketServer en précisant le port en paramètre
- définir une boucle sans fin contenant les actions ci dessous
- appelle de la méthode accept() qui renvoie une socket lors d'une nouvelle connexion
- obtenir un flux en entrée et en sortie à partir de la socket
- écrire les traitements à réaliser

Exemple (java1.2) :

```
import java.net.*;
import java.io.*;

public class TestServeurTCP {
  final static int port = 9632;

  public static void main(String[] args) {
    try {
      ServerSocket socketServeur = new ServerSocket(port);
      System.out.println("Lancement du serveur");

      while (true) {
        Socket socketClient = socketServeur.accept();
        String message = "";

        System.out.println("Connexion avec : "+socketClient.getInetAddress());

        // InputStream in = socketClient.getInputStream();
        // OutputStream out = socketClient.getOutputStream();

        BufferedReader in = new BufferedReader(
          new InputStreamReader(socketClient.getInputStream()));
        PrintStream out = new PrintStream(socketClient.getOutputStream());
        message = in.readLine();
        out.println(message);
```

```
        socketClient.close();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

L'inconvenient de ce modèle est qu'il ne peut traiter qu'une connection en même temps. Pour pouvoir traiter plusieurs connexions simultanément, il faut créer un nouveau thread contenant les traitements à réaliser sur la socket.

Exemple (java 1.2) :

```
import java.net.*;
import java.io.*;

public class TestServeurThreadTCP extends Thread {

  final static int port = 9632;
  private Socket socket;

  public static void main(String[] args) {
    try {
      ServerSocket socketServeur = new ServerSocket(port);
      System.out.println("Lancement du serveur");
      while (true) {
        Socket socketClient = socketServeur.accept();
        TestServeurThreadTCP t = new TestServeurThreadTCP(socketClient);
        t.start();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public TestServeurThreadTCP(Socket socket) {
    this.socket = socket;
  }

  public void run() {
    traitements();
  }

  public void traitements() {
    try {
      String message = "";

      System.out.println("Connexion avec le client : " + socket.getInetAddress());

      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      PrintStream out = new PrintStream(socket.getOutputStream());
      message = in.readLine();
      out.println("Bonjour " + message);

      socket.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

21.4.2. La classe Socket

Les sockets implémentent le protocole TCP (Transmission Control Protocol). La classe contient les méthodes de création des flux d'entrée et de sortie correspondants. Les sockets constituent la base des communications par le réseau.

Comme les flux Java sont transformés en format TCP/IP, il est possible de communiquer avec l'ensemble des ordinateurs qui utilisent ce même protocole. La seule chose importante au niveau du système d'exploitation est qu'il soit capable de gérer ce protocole.

Cette classe encapsule la connexion à une machine distante via le réseau. Cette classe gère la connexion, l'envoi de données, la réception de données et la déconnexion.

La classe Socket possède plusieurs constructeurs dont les principaux sont :

Constructeur	Rôle
Server()	Constructeur par défaut
ServerSocket(String, int)	Créer une socket sur la machine dont le nom et le port sont fournis en paramètre
ServerSocket(InetAddress, int)	Créer une socket sur la machine dont l'adresse IP et le port sont fournis en paramètre

La classe Socket possède de nombreuses méthodes :

Méthode	Rôle
InetAddress getInetAddress()	Renvoie l'adresse I.P. à laquelle la socket est connectée
void close()	Fermer la socket
InputStream getInputStream()	Renvoie un flux en entrée pour recevoir les données de la socket
OutputStream getOutputStream()	Renvoie un flux en sortie pour émettre les données de la socket
int getPort()	Renvoie le port utilisé par la socket

Le mise en oeuvre de la classe Socket suit toujours la même logique :

- créer une instance de la classe Socket en précisant la machine et le port en paramètre
- obtenir un flux en entrée et en sortie
- écrire les traitements à réaliser

Exemple :

```
import java.net.*;
import java.io.*;

public class TestClientTCP {
  final static int port = 9632;

  public static void main(String[] args) {

    Socket socket;
    DataInputStream userInput;
    PrintStream theOutputStream;

    try {
      InetAddress serveur = InetAddress.getByName(args[0]);
      socket = new Socket(serveur, port);

      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      PrintStream out = new PrintStream(socket.getOutputStream());

      out.println(args[1]);
      System.out.println(in.readLine());

    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

21.5. Utilisation du protocole UDP

UDP est un protocole basé sur IP qui permet une connection de type point à point ou de type multi−point. C'est un protocole qui ne garanti pas l'envoi dans l'ordre fourni des données. En contre partie, ce protocole offre de bonnes performances car il est très rapide.

C'est un protocole qui ne garantit pas la transmission correcte des données et qui devrait donc être réservé à des taches peu importantes. Ce protocole est en revanche plus rapide que le protocole TCP.

Pour assurer les échanges, UDP utilise la notion de port, ce qui permet à plusieurs applications d'utiliser UDP sans que les échanges interfairent les uns avec les autres. Cette notion est similaire à la notion de port utilisé par TCP.

UDP est utilisé dans de nombreux services "standards" tel que echo (port 7), DayTime (13), etc ...

L'échange de données avec UDP se fait avec deux sockets, l'une sur le serveur, l'autre sur le client. Chaque socket est caractérisée par une adresse internet et un port.

Pour utiliser le protocole UDP, java défini deux classes DatagramSocket et DatagramPacket.

21.5.1. La classe DatagramSocket

Cette classe crée un Socket qui utilise le protocole UDP (Unreliable Datagram Protocol) pour émettre ou recevoir des données.

Cette classe possède plusieurs constructeurs :

Constructeur	Rôle
DatagramSocket()	Créé une socket attachée à toutes les adresses IP de la machine et un à des ports libres sur la machine
DatagramSocket(int)	Créé une socket attachée à toutes les adresses IP de la machine et au port précisé en paramètre
DatagramSocket(int, InetAddress)	Créé une socket attachée à adresse IP et au port précisé en paramètre

Tous les constructeurs peuvent lever une exception de type SocketException : en particulier, si le port précisé est déjà utilisé lors de l'instanciation de l'objet DatagramSocket, une exception de type BindException est levée. Cette exception hérite de SocketException.

La classe DatagramSocket définit plusieurs méthodes :

Méthode	Rôle
close()	Fermer la Socket et ainsi libérer le port
receive(DatagramPacket)	Recevoir des données
send(DatagramPacket)	Envoyer des données
int getPort()	Renvoie le port associé à la socket
void setSoTimeout(int)	Préciser un timeout d'attente pour la réception d'un message.

Par défaut, un objet DatagramSocket ne possède pas de timeout lors de l'utilisation de la méthode receive(). La méthode bloque donc l'exécution de l'application jusqu'à la réception d'un packet de données. La méthode setSoTimeout() permet de préciser un timeout en millisecondes. Une fois ce délai écoulé sans réception d'un paquet de données, la méthode lève une exception du type SocketTimeoutException.

21.5.2. La classe DatagramPacket

Cette classe encapsule une adresse internet, un port et les données qui sont échangées grâce à un objet de type DatagramSocket. Elle possède plusieurs constructeurs pour encapsuler des paquets émis ou reçus.

Constructeur	Rôle
DatagramPacket(byte tampon[], int taille)	Encapsule des paquets en réception dans un tampon
DatagramPacket(byte port[], int taille, InetAddress adresse, int port)	Encapsule des paquets en émission à destination d'une machine

Cette classe propose plusieurs méthodes pour obtenir ou mettre à jour les informations sur le paquet encapsulé.

Méthode	Rôle
InetAddress getAddress ()	Renvoie l'adresse du serveur
byte[] getData()	Renvoie les données contenues dans le paquet
int getPort ()	Renvoie le port
int getLength ()	Renvoie la taille des données contenues dans le paquet
setData(byte[])	Mettre à jour les données contenues dans le paquet

Le format des données échangées est un tableau d'octets, il faut donc correctement initialiser la propriété length qui représente la taille du tableau pour un paquet émis et utiliser cette propriété pour lire les données dans un paquet reçu.

21.5.3. Un exemple de serveur et de client

L'exemple suivant est très simple : un serveur attend un nom d'utilisateur envoyé sur le port 9632. Dès qu'un message lui est envoyé, il renvoie à son expéditeur "bonjour" suivi du nom envoyé.

Exmple : le serveur

```
import java.io.*;
import java.net.*;

public class TestServeurUDP {

  final static int port = 9632;
  final static int taille = 1024;
  static byte buffer[] = new byte[taille];

  public static void main(String argv[]) throws Exception {
    DatagramSocket socket = new DatagramSocket(port);
    String donnees = "";
    String message = "";
    int taille = 0;

    System.out.println("Lancement du serveur");
    while (true) {
      DatagramPacket paquet = new DatagramPacket(buffer, buffer.length);
      DatagramPacket envoi = null;
      socket.receive(paquet);

      System.out.println("\n"+paquet.getAddress());
      taille = paquet.getLength();
      donnees = new String(paquet.getData(),0, taille);
      System.out.println("Donnees reçues = "+donnees);

      message = "Bonjour "+donnees;
```

```
      System.out.println("Donnees envoyees = "+message);
      envoi = new DatagramPacket(message.getBytes(),
        message.length(), paquet.getAddress(), paquet.getPort());
      socket.send(envoi);
    }
  }
}
```

Exemple : le client

```
import java.io.*;
import java.net.*;

public class TestClientUDP {

  final static int port = 9632;
  final static int taille = 1024;
  static byte buffer[] = new byte[taille];

  public static void main(String argv[]) throws Exception {
    try {
      InetAddress serveur = InetAddress.getByName(argv[0]);
      int length = argv[1].length();
      byte buffer[] = argv[1].getBytes();
      DatagramSocket socket = new DatagramSocket();
      DatagramPacket donneesEmises = new DatagramPacket(buffer, length, serveur, port);
      DatagramPacket donneesRecues = new DatagramPacket(new byte[taille], taille);

      socket.setSoTimeout(30000);
      socket.send(donneesEmises);
      socket.receive(donneesRecues);

      System.out.println("Message : " + new String(donneesRecues.getData(),
        0, donneesRecues.getLength()));
      System.out.println("de : " + donneesRecues.getAddress() + ":" +
        donneesRecues.getPort());
    } catch (SocketTimeoutException ste) {
      System.out.println("Le delai pour la reponse a expire");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Pour utiliser le client, il faut connaitre l'adresse internet de la machine sur laquelle le serveur. L'appel du client nécessite de fournir en paramètre l'adresse internet du serveur et le nom de l'utilisateur.

Exécution du client

```
C:\>java TestClientUDP www.test.fr "Michel"
java.net.UnknownHostException: www.test.fr: www.test.fr
        at java.net.InetAddress.getAllByName0(InetAddress.java:948)
        at java.net.InetAddress.getAllByName0(InetAddress.java:918)
        at java.net.InetAddress.getAllByName(InetAddress.java:912)
        at java.net.InetAddress.getByName(InetAddress.java:832)
        at TestClientUDP.main(TestClientUDP.java:12)

C:\>java TestClientUDP 192.168.25.101 "Michel"
Le delai pour la reponse a expire

C:\>java TestClientUDP 192.168.25.101 "Michel"
Message : Bonjour Michel
de : /192.168.25.101:9632
```

21.6. Les exceptions liées au réseau

Le package java.net définit plusieurs exceptions :

Exception	
BindException	Connection au port local impossible : le port est peut être déjà utilisé
ConnectException	Connection à une socket impossible : aucun serveur n'écoute sur le port précisé
MalformedURLException	L'URL n'est pas valide
NoRouteToHostException	Connection à l'hote impossible : un firewall empeche la connexion
ProtocolException	
SocketException	
SocketTimeoutException	Délai d'attente pour le reception ou l'émission des données écoulé
UnknownHostException	L'adresse IP de l'hote n'a pas pu être trouvée
UnknownServiceException	
URISyntaxException	

21.7. Les interfaces de connexions au réseau

Le J2SE 1.4 ajoute une nouvelle classe qui encapsule une interface de connexion au réseau et qui permet d'obtenir la liste des interfaces de connexion au réseau de la machine. Cette classe est la classe NetworkInterface.

Une interface de connexion au réseau se caractérise par un nom court, une désignation et une liste d'adresses IP. La classe possède des getters sur chacun de ces éléments :

Méthode	Rôle
String getName()	Renvoie le nom court de l'interface
String getDisplayName()	Renvoie la désignation de l'interface
Enumeration getInetAddresses()	Renvoie une énumération d'objet InetAddress contenant la liste des adresses IP associée à l'interface

Cette classe possède une méthode statique getNetwotkInterfaces() qui renvoie une énumeration contenant des objets de type NetworkInterface encapsulant les différentes interfaces présentes dans la machine.

Exemple :

```
import java.net.*;
import java.util.*;

public class TestNetworkInterface {

  public static void main(String[] args) {
    try {
      TestNetworkInterface.getLocalNetworkInterface();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  private static void getLocalNetworkInterface() throws SocketException, NoClassDefFoundError {
    Enumeration interfaces = NetworkInterface.getNetworkInterfaces();
```

```
    while (interfaces.hasMoreElements()) {
      NetworkInterface ni;
      Enumeration adresses;

      ni = (NetworkInterface) interfaces.nextElement();

      System.out.println("Network interface : ");
      System.out.println("  nom court    = " + ni.getName());
      System.out.println("  désignation  = " + ni.getDisplayName());

      adresses = ni.getInetAddresses();
      while (adresses.hasMoreElements()) {
        InetAddress ia = (InetAddress) adresses.nextElement();
        System.out.println("  adresse I.P. = " + ia);
      }
    }
  }

}
```

Résultat :

```
Network interface :
  nom court    = MS TCP Loopback interface
  désignation  = lo
  adresse I.P. = /127.0.0.1
Network interface :
  nom court    = Carte Realtek Ethernet à base RTL8029(AS)(Générique)
  désignation  = eth0
  adresse I.P. = /169.254.166.156
Network interface :
  nom court    = WAN (PPP/SLIP) Interface
  désignation  = ppp0
  adresse I.P. = /193.251.70.245
```

22. L'accès aux bases de données : JDBC

JDBC est l'acronyme de Java DataBase Connectivity et désigne une API définie par Sun pour permettre un accès aux bases de données avec Java.

Ce chapitre présente dans plusieurs sections l'utilisation de cette API :

22.1. Les outils nécessaires pour utiliser JDBC

Les classes de JDBC version 1.0 sont regroupées dans le package java.sql et sont incluses dans le JDK à partir de sa version 1.1. La version 2.0 de cette API est incluse dans la version 1.2 du JDK.

Pour pouvoir utiliser JDBC, il faut un pilote qui est spécifique à la base de données à laquelle on veut accéder. Avec le JDK, Sun fournit un pilote qui permet l'accès aux bases de données via ODBC.

Ce pilote permet de réaliser l'indépendance de JDBC vis à vis des bases de données.

Pour utiliser le pont JDBC−ODBC sous Window 9x, il faut utiliser ODBC en version 32 bits.

22.2. Les types de pilotes JDBC

Il existe quatre types de pilote JDBC :

1. Type 1 (JDBC−ODBC bridge) : le pont JDBC−ODBC qui s'utilise avec ODBC et un pilote ODBC spécifique pour la base à accéder. Cette solution fonctionne très bien sous Windows. C'est la solution idéale pour des développements avec exécution sous Windows d'une application locale. Cette solution « simple » pour le développement possède plusieurs inconvenients :
    - ♦ la multiplication du nombre de couches rend complexe l'architecture (bien que transparent pour le développeur) et détériore un peu les performances

- lors du deploiement, ODBC et son pilote doivent être installé sur tous les postes ou l'application va fonctionner.
- la partie native (ODBC et son pilote) rend l'application moins portable et dépendant d'une plateforme.

2. Type 2 : un driver écrit en java qui appelle l'API native de la base de données

   Ce type de driver convertit les ordres JDBC pour appeler directement les API de la base de données via un pilote natif sur le client. Ce type de driver nécessite aussi l'utilisation de code natif sur le client.

3. Type 3 : un driver écrit en Java utilisant le protocole natif de la base de données

   Ce type de driver utilise un protocole réseau propriétaire spécifique à une base de données. Un serveur dédié reçoit les messages par ce protocole et dialogue directement avec la base de données. Ce type de driver peut être facilement utilisé par une applet mais dans ce cas le serveur intermédiaire doit obligatoirement être installé sur la machine contenant le serveur web.

4. Type 4 : un driver Java natif

   Ce type de driver, écrit en java, appelle directement le SGBD par le réseau. Ils sont fournis par l'éditeur de la base de données.

Les drivers se présentent souvent sous forme de fichiers jar dont le chemin doit être ajouté au classpath pour permettre au programme de l'utiliser.

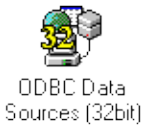
22.3. Enregistrer une base de données dans ODBC sous Windows 9x ou XP

Pour utiliser un pilote de type 1 (pont ODBC−JDBC) sous Windows 9x, il est nécessaire d'enregistrer la base de données dans ODBC avant de pouvoir l'utiliser.

Attention : ODBC n'est pas fourni en standard avec Windows 9x.

Pour enregistrer une nouvelle base de données, il faut utiliser l'administrateur de source de données ODBC.

Pour lancer cette application sous Windows 9x, il faut doubler cliquer sur l'icône "ODBC 32bits" dans le panneau de configuration.



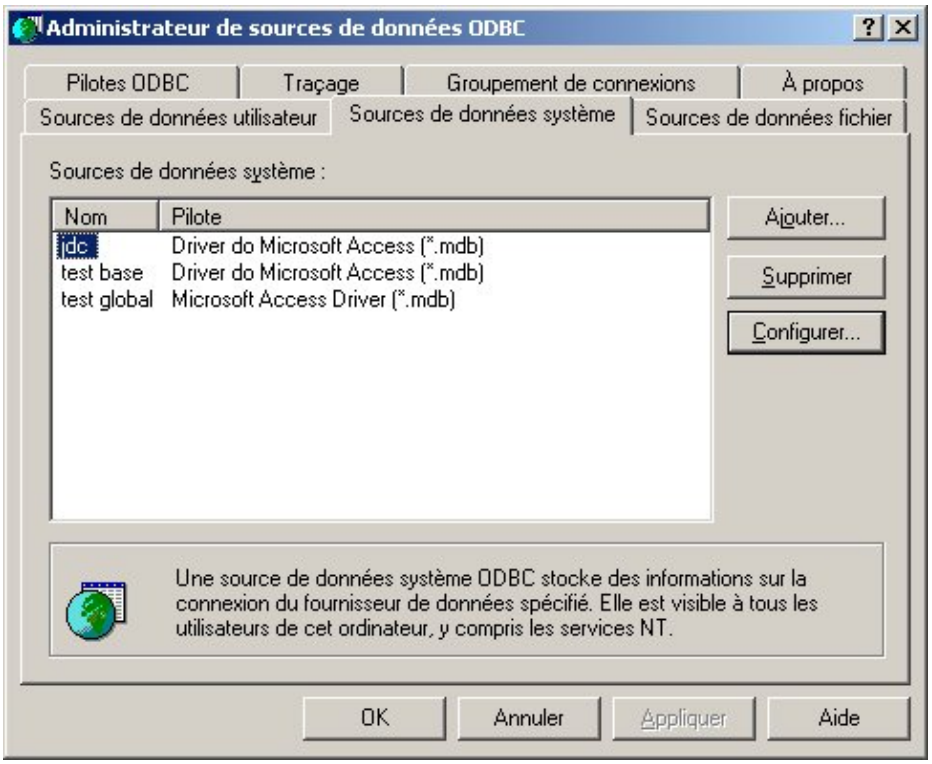
Sous Windows XP, il faut double cliquer sur l'icône "Source de données (ODBC)" dans le répertoire "Outils d'administration" du panneau de configuration.

L'outil se compose de plusieurs onglets.

L'onglet "Pilote ODBC" liste l'ensemble des pilotes qui sont installés sur la machine.

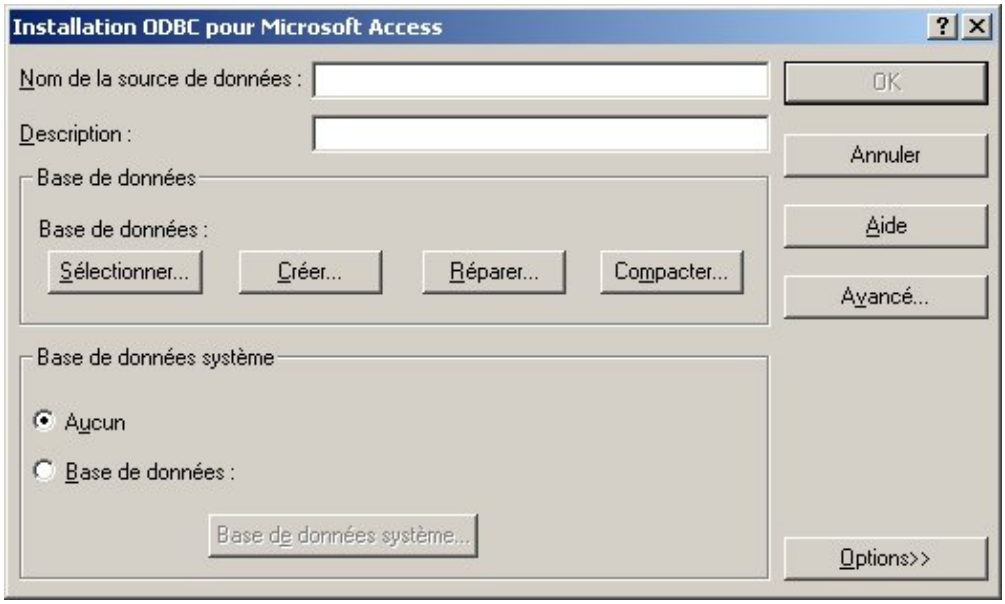
L'onglet "Source de données utilisateur" liste l'ensemble des sources de données pour l'utilisateur courament connecté sous Windows.

L'onglet "Source de données système" liste l'ensemble des sources de données accessibles par tous les utilisateurs.



Le plus simple est de créer une telle source de données en cliquant sur le bouton "Ajouter". Une boite de dialogue permet de sélectionner le pilote qui sera utilisé par la source de donées.

Il suffit de sélectionner le pilote et de cliquer sur "Terminer". Dans l'exemple ci dessous, le pilote sélectionné concerne une base Microsoft Access.



Il suffit de saisir les informations nécessaires notamment le nom de la source de données et de sélectionner la base. Un clic sur le bouton "Ok" crée la source de données qui pourra alors être utilisée.

22.4. Présentation des classes de l'API JDBC

Toutes les classes de JDBC sont dans le package java.sql. Il faut donc l'importer dans tous les programmes devant utiliser JDBC.

Exemple (code Java 1.1) :
`import java.sql.*;`

Il y a 4 classes importantes : DriverManager, Connection, Statement (et PreparedStatement), et ResultSet, chacune correspondant à une étape de l'accès aux données :

Classe	Role
DriverManager	charge et configure le driver de la base de données.
Connection	réalise la connection et l'authentification à la base de données.
Statement (et PreparedStatement)	contient la requête SQL et la transmet à la base de données.
ResultSet	permet de parcourir les informations retournées par la base de données dans le cas d'une sélection de données

Chacunes de ces classes dépend de l'instanciation d'un objet de la précédente classe.

22.5. La connexion à une base de données

22.5.1. Le chargement du pilote

Pour se connecter à une base de données via ODBC, il faut tout d'abord charger le pilote JDBC−ODBC qui fait le lien entre les deux.

Exemple (code Java 1.1) :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Pour se connecter à une base en utilisant un driver spécifique, la documentation du driver fournit le nom de la classe à utiliser. Par exemple, si le nom de la classe est jdbc.DriverXXX, le chargement du driver se fera avec le code suivant :

Class.forName("jdbc.DriverXXX");

Exemple (code Java 1.1) : Chargement du pilote pour un base PostgreSQL sous Linux

```
Class.forName("postgresql.Driver");
```

Il n'est pas nécessaire de créer une instance de cette classe et de l'enregistrer avec le DriverManager car l'appel à Class.forName le fait automatiquement : ce traitement charge le pilote et créer une instance de cette classe.

La méthode static forName() de la classe Class peut lever l'exception java.lang.ClassNotFoundException.

22.5.2. L'établissement de la connection

Pour se connecter à une base de données, il faut instancier un objet de la classe Connection en lui précisant sous forme d'URL la base à accéder.

Exemple (code Java 1.1) : Etablir une connexion sur la base testDB via ODBC

```
String DBurl = "jdbc:odbc:testDB";

con = DriverManager.getConnection(DBurl);
```

La syntaxe URL peut varier d'un type de base de données à l'autre mais elle est toujours de la forme : protocole:sous_protocole:nom

« jbdc » désigne le protocole est vaut toujours « jdbc ». « odbc » désigne le sous protocole qui définit le mécanisme de connection pour un type de bases de données.

Le nom de la base de données doit être celui saisi dans le nom de la source sous ODBC.

La méthode getConnection() peut lever une exception de la classe java.sql.SQLException.

Le code suivant décrit la création d'une connection avec un user et un mot de passe :

Exemple (code Java 1.1) :

```
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

A la place de " myLogin " ; il faut mettre le nom du user qui se connecte à la base et mettre son mot de passe à la place de "myPassword "

Exemple (code Java 1.1) :

```
String url = "jdbc:odbc:factures";

Connection con = DriverManager.getConnection(url, "toto", "passwd");
```

La documentation d'un autre driver indiquera le sous protocole à utiliser (le protocole à mettre derrière jdbc dans l'URL).

Exemple (code Java 1.1) : Connection à la base PostgreSQL nommée test avec le user jumbo et le mot de passe 12345 sur la machine locale

```
Connection con=DriverManager.getConnection("jdbc:postgresql://localhost/test","jumbo","12345");
```

22.6. Accéder à la base de données

Une fois la connection établie, il est possible d'exécuter des ordres SQL. Les objets qui peuvent être utilisés pour obtenir des informations sur la base de données sont :

Classe	Role
DatabaseMetaData	informations à propos de la base de données : nom des tables, index, version ...
ResultSet	résultat d'une requête et information sur une table. L'accès se fait enregistrement par enregistrement.
ResultSetMetaData	informations sur les colonnes (nom et type) d'un ResultSet

22.6.1. L'execution de requêtes SQL

Les requêtes d'interrogation SQL sont exécutées avec les méthodes d'un objet Statement que l'on obtient à partir d'un objet Connection

Exemple (code Java 1.1) :

```
ResultSet résultats = null;
String requete = "SELECT * FROM client";

try {
   Statement stmt = con.createStatement();
   résultats = stmt.executeQuery(requete);
} catch (SQLException e) {
   //traitement de l'exception
}
```

Un objet de la classe Statement permet d'envoyer des requetes SQL à la base. Le création d'un objet Statement s'effectue à partir d'une instance de la classe Connection :

Exemple (code Java 1.1) :

```
Statement stmt = con.createStatement();
```

Pour une requête de type interrogation (SELECT), la méthode à utiliser de la classe Statement est exécuteQuery. Pour des traitements de mise à jour, il faut utiliser la méthode executeUpdate(). Lors de l'appel à la méthode d'exécution, il est nécessaire de lui fournir en paramètre la requête SQL sous forme de chaine.

Le résultat d'une requête d'intérrogation est renvoyé dans un objet de la classe ResultSet par la méthode executeQuery().

Exemple (code Java 1.1) :

```
ResultSet rs = stmt.executeQuery("SELECT * FROM employe");
```

La méthode executeUpdate() retourne le nombre d'enregistrements qui ont été mis à jour

Exemple (code Java 1.1) :

```
...

//insertion d'un enregistrement dans la table client

requete = "INSERT INTO client VALUES (3,'client 3','prenom 3')";
try {
   Statement stmt = con.createStatement();
   int nbMaj = stmt.executeUpdate(requete);
   affiche("nb mise a jour = "+nbMaj);
} catch (SQLException e) {
   e.printStackTrace();
}

...
```

Lorsque la méthode executeUpdate() est utilisée pour exécuter un traitement de type DDL (Data Defiition Langage : définition de données) comme la création d'un table, elle retourne 0. Si la méthode retourne 0, cela peut signifier deux choses : le traitement de mise à jour n'a affecté aucun enregistrement ou le traitement concernait un traitement de type DDL.

Si l'on utilise executeQuery() pour exécuter une requête SQL ne contenant pas d'ordre SELECT, alors une exception de type SQLException est levée.

Exemple (code Java 1.1) :

```
...

requete = "INSERT INTO client VALUES (4,'client 4','prenom 4')";
try {
   Statement stmt = con.createStatement();
   ResultSet résultats = stmt.executeQuery(requete);
} catch (SQLException e) {
   e.printStackTrace();
}

...
```

résultat :

```
java.sql.SQLException: No ResultSet was produced
java.lang.Throwable(java.lang.String)
java.lang.Exception(java.lang.String)
java.sql.SQLException(java.lang.String)
java.sql.ResultSet sun.jdbc.odbc.JdbcOdbcStatement.executeQuery(java.lang.String)
void testjdbc.TestJDBC1.main(java.lang.String [])
```

Attention : dans ce cas la requête est quand même effectuée. Dans l'exemple, un nouvel enregistrement est créé dans la table.

Il n'est pas nécessaire de définir un objet Statement pour chaque ordre SQL : il est possible d'un définir un et de le réutiliser

22.6.2. La classe ResultSet

C'est une classe qui représente une abstraction d'une table qui se compose de plusieurs enregistrements constitués de colonnes qui contiennent les données.

Les principales méthodes pour obtenir des données sont :

Méthode	Role

getInt(int)	retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme d'entier.
getInt(String)	retourne le contenu de la colonne dont le nom est passé en paramètre sous forme d'entier.
getFloat(int)	retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme de nombre flottant.
getFloat(String)	
getDate(int)	retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme de date.
getDate(String)	
next()	se déplace sur le prochain enregistrement : retourne false si la fin est atteinte
Close()	ferme le ResultSet
getMetaData()	retourne un objet ResultSetMetaData associé au ResultSet.

La méthode getMetaData() retourne un objet de la classe ResultSetMetaData qui permet d'obtenir des informations sur le résultat de la requête. Ainsi, le nombre de colonne peut être obtenu grace à la méthode getColumnCount de cet objet.

Exemple (code Java 1.1) :

```
ResultSetMetaData rsmd;
rsmd = results.getMetaData();
nbCols = rsmd.getColumnCount();
```

La méthode next() déplace le curseur sur le prochain enregistrement. Le curseur pointe initialement juste avant le premier enregistrement : il est nécessaire de faire un premier appel à la méthode next() pour se placer sur le premier enregistrement.

Des appels successifs à next permettent de parcourir l'ensemble des enregistrements.

Elle retourne false lorsqu'il n'y a plus d'enregistrement. Il faut toujours protéger le parcours d'une table dans un bloc de capture d'exception

Exemple (code Java 1.1) :

```
//parcours des données retournées

try {
   ResultSetMetaData rsmd = résultats.getMetaData();
   int nbCols = rsmd.getColumnCount();
   boolean encore = résultats.next();
   while (encore) {
      for (int i = 1; i <= nbCols; i++)
         System.out.print(résultats.getString(i) + " ");
      System.out.println();
      encore = résultats.next();
   }
   résultats.close();
} catch (SQLException e) {
   //traitement de l'exception
}
```

Les méthodes getXXX() permettent d'extraire les données selon leur type spécifiée par XXX tel que getString(), getDouble(), getInteger(), etc ... . Il existe deux formes de ces méthodes : indiquer le numéro la colonne en paramètre (en commençant par 1) ou indiquer le nom de la colonne en paramètre. La première méthode est plus efficace mais peut générer plus d'erreurs à l'éxecution notamment si la structure de la table évolue.

Attention : il est important de noter que ce numéro de colonne fourni en paramètre fait référence au numéro de colonne de l'objet resultSet (celui correspondant dans l'ordre SELECT)et non au numéro de colonne de la table.

La méthode getString() permet d'obtenir la valeur d'un champ de n'importe quel type.

22.6.3. Exemple complet de mise à jour et de sélection sur une table

Exemple (code Java 1.1) :

```
import java.sql.*;

public class TestJDBC1 {

   private static void affiche(String message) {
      System.out.println(message);
   }

   private static void arret(String message) {
      System.err.println(message);
      System.exit(99);
   }

   public static void main(java.lang.String[] args) {
      Connection con = null;
      ResultSet résultats = null;
      String requete = "";

      // chargement du pilote
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      } catch (ClassNotFoundException e) {
         arret("Impossible de charger le pilote jdbc:odbc");
      }

      //connection a la base de données

      affiche("connection a la base de données");
      try {

         String DBurl = "jdbc:odbc:testDB";
         con = DriverManager.getConnection(DBurl);
      } catch (SQLException e) {
         arret("Connection à la base de données impossible");
      }

      //insertion d'un enregistrement dans la table client
      affiche("creation enregistrement");

      requete = "INSERT INTO client VALUES (3,'client 3','client 4')";
      try {
         Statement stmt = con.createStatement();
         int nbMaj = stmt.executeUpdate(requete);
         affiche("nb mise a jour = "+nbMaj);
      } catch (SQLException e) {
          e.printStackTrace();
      }

      //creation et execution de la requete
      affiche("creation et execution de la requête");
      requete = "SELECT * FROM client";

      try {
         Statement stmt = con.createStatement();
         résultats = stmt.executeQuery(requete);
      } catch (SQLException e) {
         arret("Anomalie lors de l'execution de la requête");
      }
```

```
      //parcours des données retournées
      affiche("parcours des données retournées");
      try {
         ResultSetMetaData rsmd = résultats.getMetaData();
         int nbCols = rsmd.getColumnCount();
         boolean encore = résultats.next();

         while (encore) {

            for (int i = 1; i <= nbCols; i++)
               System.out.print(résultats.getString(i) + " ");
            System.out.println();
            encore = résultats.next();
         }

         résultats.close();
      } catch (SQLException e) {
         arret(e.getMessage());
      }

      affiche("fin du programme");
      System.exit(0);
   }
}
```

résultat :

```
connection a la base de données
creation enregistrement
nb mise a jour = 1
creation et execution de la requête
parcours des données retournées
1.0 client 1 prenom 1
2.0 client 2 prenom 2
3.0 client 3 client 4
fin du programme
```

22.7. Obtenir des informations sur la base de données

22.7.1. La classe ResultSetMetaData

La méthode getMetaData() d'un objet ResultSet retourne un objet de typeResultSetMetaData. Cet objet permet de connaître le nombre, le nom et le type des colonnes.

Méthode	Role
int getColumnCount()	retourne le nombre de colonnes du ResultSet
String getColumnName(int)	retourne le nom de la colonne dont le numéro est donné
String getColumnLabel(int)	retourne le libellé de la colonne donnée
boolean isCurrency(int)	retourne true si la colonne contient un nombre au format monétaire
boolean isReadOnly(int)	retourne true si la colonne est en lecture seule
boolean isAutoIncrement(int)	retourne true si la colonne est auto incrémentée
int getColumnType(int)	retourne le type de données SQL de la colonne

22.7.2. La classe DatabaseMetaData

Un objet de la classe DatabaseMetaData permet d'obtenir des informations sur la base de données dans son ensemble : nom des tables, nom des colonnes dans une table, méthodes SQL supportées

Méthode	Role
ResultSet getCatalogs()	retourne la liste du catalogue d'informations (Avec le pont JDBC−ODBC, on obtient la liste des bases de données enregistrées dans ODBC).
ResultSet getTables(catalog, schema, tableNames, columnNames)	retourne une description de toutes les tables correspondant au TableNames donné et à toutes les colonnes correspondantes à columnNames.
ResultSet getColumns(catalog, schema, tableNames, columnNames)	retourne une description de toutes les colonnes correspondantes au TableNames donné et à toutes les colonnes correspondantes à columnNames.
String getURL()	retourne l'URL de la base à laquelle on est connecté
String getDriverName()	retourne le nom du driver utilisé

La méthode getTables() de l'objet DataBaseMetaData demande quatre arguments :

getTables(catalog, schema, tablemask, types[]);

- catalog : le nom du catalogue dans lequel les tables doivent être recherchées. Pour une base de données JDBC−ODBC, il peut être mis à null.
- schema : le schéma de la base données à inclure pour les bases les supportant. Il est en principe mis à null
- tablemask : un masque décrivant les noms des tables à retrouver. Pour les retrouver toutes, il faut l'initialiser avec la caractères '%'
- types[] : tableau de chaines décrivant le type de tables à retrouver. La valeur null permet de retrouver toutes les tables.

Exemple (code Java 1.1) :

```
con = DriverManager.getConnection(url);
dma =con.getMetaData();
String[] types = new String[1];
types[0] = "TABLES"; //set table type mask

results = dma.getTables(null, null, "%", types);

boolean more = results.next();
while (more) {
   for (i = 1; i <= numCols; i++)
      System.out.print(results.getString(i)+" ");
   System.out.println();
   more = results.next();
}
```

22.8. L'utilisation d'un objet PreparedStatement

L'interface PreparedStatement définit les méthodes pour un objet qui va encapsuler une requête pré−compilée. Ce type de requête est particulièrement adapté pour une exécution répétée d'une même requête avec des paramètres différents.

Cette interface hérite de l'interface Statement.

Lors de l'utilisation d'un objet de type PreparedStatement, la requête est envoyée au moteur de la base de données pour que celui ci prépare son exécution.

Un objet qui implémente l'interface PreparedStatement est obtenu en utilisant la méthode preparedStatement() d'un objet de type Connection. Cette méthode attend en paramètre une chaîne de caractères contenant la requête SQL. Dans cette chaine, chaque paramètre est représenté par un caractère ?.

Un ensemble de méthode setXXX() (ou XXX représente un type primitif ou certains objets tel que String, Date, Object, ...) permet de fournir les valeurs de chaque paramètre défini dans la requête. Le premier paramètre de ces méthodes précise le numéro du paramètre dont la méthode va fournir la valeur. Le second paramètre précise cette valeur.

Exemple (code Java 1.1) :

```
package com.jmd.test.dej;

import java.sql.*;

public class TestJDBC2 {

  private static void affiche(String message) {
    System.out.println(message);
  }

  private static void arret(String message) {
    System.err.println(message);
    System.exit(99);
  }

  public static void main(java.lang.String[] args) {
    Connection con = null;
    ResultSet resultats = null;
    String requete = "";

    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException e) {
      arret("Impossible de charger le pilote jdbc:odbc");
    }

    affiche("connection a la base de données");
    try {

      String DBurl = "jdbc:odbc:testDB";
      con = DriverManager.getConnection(DBurl);
      PreparedStatement recherchePersonne =
        con.prepareStatement("SELECT * FROM personnes WHERE nom_personne = ?");

      recherchePersonne.setString(1, "nom3");

      resultats = recherchePersonne.executeQuery();

      affiche("parcours des données retournées");

      boolean encore = resultats.next();

      while (encore) {
        System.out.print(resultats.getInt(1) + " :  "+resultats.getString(2)+" "+
          resultats.getString(3)+"("+resultats.getDate(4)+")");
        System.out.println();
        encore = resultats.next();
      }

      resultats.close();
    } catch (SQLException e) {
      arret(e.getMessage());
    }

    affiche("fin du programme");
    System.exit(0);
  }
}
```

Pour exécuter la requête, l'interface PreparedStatement propose deux méthodes :

- executeQuery() : cette méthode permet d'exécuter une requête de type interrogation et renvoie un objet de type ResultSet qui contient les données issues de l'exécution de la requête
- executeUpdate() : cette méthode permet d'exécuter une requête de type mise à jour et renvoie un entier qui contient le nombre d'occurences impactées par la mise à jour

22.9. L'utilisation des transactions

Une transaction permet de ne valider un ensemble de traitements sur le base de données que si ils se sont tous effectués correctement.

Par exemple, une opération bancaire de transfert de fond d'un compte vers un autre oblige à la réalisation de l'opération de débit sur un compte et de l'opération de crédit sur l'autre compte. La réalisation d'une seule de ces opérations laisserait les données de la base dans un état inconsistant.

Une transaction est un mécanisme qui permet donc de s'assurer que toutes les opérations qui la compose seront réellement effectuées.

Une transaction est gérée à partir de l'objet Connection. Par défaut, une connection est en mode auto−commit. Dans ce mode, chaque opération est validée unitairement pour former la transaction.

Pour pouvoir rassembler plusieurs traitements dans une transaction, il faut tout d'abord désactiver le mode auto−commit. La classe Connection possède la méthode setAutoCommit() qui attend un booléen qui précise le mode de fonctionnement.

Exemple :

```
connection.setAutoCommit(false);
```

Une fois le mode auto−commit désactivé, un appel à la méthode commit() de la classe Connection permet de valider la transaction courante. L'appel à cette méthode valide la transaction courante et créé implicitement une nouvelle transaction.

Si une anomalie intervient durant la transaction, il est possible de faire un retour en arrière pour revenir à la situation de la base de données au début de la transaction en appellant la méthode rollback() de la classe Connection.

22.10. Les procédures stockées

L'interface CallableStatement définit les méthodes pour un objet qui va permettre d'appeler une procédure stockée.

Cette interface hérite de l'interface PreparedStatement.

Un objet qui implémente l'interface CallableStatement est obtenu en utilisant la méthode prepareCall() d'un objet de type Connection. Cette méthode attend en paramètre une chaîne de caractères contenant la chaîne d'appel de la procédure stockée.

L'appel d'une procédure étant particulier à chaque base de données supportant une telle fonctionnalité, JDBC propose une syntaxe unifiée qui sera transcrite par le pilote en un appel natif à la base de données. Cette syntaxe peut prendre plusieurs formes :

- {call nom_procedure_stockees} : cette forme la plus simple permet l'appel d'une procédure stockée sans paramètre ni valeur de retour
- {call nom_procedure_stockees(?, ?, ...)} : cette forme permet l'appel d'une procédure stockée avec des paramètres
- {? = call nom_procedure_stockees(?, ?, ...)} : cette forme permet l'appel d'une procédure stockée avec des paramètre et une valeur de retour

Un ensemble de méthode setXXX() (ou XXX représente un type primitif ou certains objets tel que String, Date, Object, ...) permet de fournir les valeurs de chaque paramètre défini dans la requête. Le premier paramètre de ces méthodes précise le numéro du paramètre dont la méthode va fournir la valeur. Le second paramètre précise cette valeur.

Un ensemble de méthode getXXX() (ou XXX représente un type primitif ou certains objets tel que String, Date, Object, ...) permet d'obtenir la valeur du paramètre de retour en fournissant la valeur 0 comme index de départ et un autre index pour les paramètres définis en entrée/sortie dans la procédure stockée.

Pour exécuter la requête, l'interface PreparedStatement propose deux méthodes :

- executeQuery() : cette méthode permet d'exécuter une requête de type interrogation et renvoie un objet de type ResultSet qui contient les données issues de l'exécution de la requête
- executeUpdate() : cette méthode permet d'exécuter une requête de type mise à jour et renvoie un entier qui contient le nombre d'occurences impactées par la mise à jour

22.11. Le traitement des erreurs JDBC

JDBC permet de connaitre les avertissements et les exceptions générées par la base de données lors de l'exécution de requête.

La classe SQLException représente les erreurs émises par la base de données. Elle contient trois attributs qui permettent de préciser l'erreur :

- message : contient une description de l'erreur
- SQLState : code défini par les normes X/Open et SQL99
- ErrorCode : le code d'erreur du fournisseur du pilote

La classe SQLException possède une méthode getNextException() qui permet d'obtenir les autres exceptions levées durant la requête. La méthode renvoie null une fois la dernière exception renvoyée.

Exemple (code Java 1.1) :

```
package com.jmd.test.dej;

import java.sql.*;

public class TestJDBC3 {

  private static void affiche(String message) {
    System.out.println(message);
  }

  private static void arret(String message) {
    System.err.println(message);
    System.exit(99);
  }

  public static void main(java.lang.String[] args) {
    Connection con = null;
    ResultSet resultats = null;
    String requete = "";

    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException e) {
      arret("Impossible de charger le pilote jdbc:odbc");
    }

    affiche("connection a la base de données");
    try {

      String DBurl = "jdbc:odbc:testDB";
      con = DriverManager.getConnection(DBurl);

      requete = "SELECT * FROM tableinexistante";
```

```
      Statement stmt = con.createStatement();
      resultats = stmt.executeQuery(requete);

      affiche("parcours des données retournées");

      boolean encore = resultats.next();

      while (encore) {
        System.out.print(resultats.getInt(1) + " :  " + resultats.getString(2) +
          " " + resultats.getString(3) + "(" + resultats.getDate(4) + ")");
        System.out.println();
        encore = resultats.next();
      }

      resultats.close();
    } catch (SQLException e) {
      System.out.println("SQLException");
      do {
        System.out.println("SQLState : " + e.getSQLState());
        System.out.println("Description :  " + e.getMessage());
        System.out.println("code erreur :   " + e.getErrorCode());
        System.out.println("");
        e = e.getNextException();
      } while (e != null);
      arret("");
    } catch (Exception e) {
      e.printStackTrace();
    }

    affiche("fin du programme");
    System.exit(0);
  }
}
```

22.12. JDBC 2.0

La version 2.0 de l'API JDBC a été intégrée au JDK 1.2. Cette nouvelle version apporte plusieurs fonctionnalités très intéréssantes dont les principales sont :

- support du parcours dans les deux sens des résultats
- support de la mise à jour des résultats
- possibilité de faire des mises à jour de masse (Batch Updates)
- prise en compte des champs définis par SQL−3 dont BLOB et CLOB

L'API JDBC 2.0 est séparée en deux parties :

- la partie principale (core API) contient les classes et interfaces nécessaires à l'utilisation de bases de données : elles sont regroupées dans le package java.sql
- la seconde partie est une extension utilisée dans J2EE qui permet de gérer les transactions distribuées, les pools de connection, la connection avec un objet DataSource ... Les classes et interfaces sont regroupées dans le package javax.sql

22.12.1. Les fonctionnalités de l'objet ResultSet

Les possibilités de l'objet ResultSet dans la version 1.0 de JDBC sont très limitées : parcours séquentiel de chaque occurence de la table retournée.

La version 2.0 apporte de nombreuses améliorations à cet objet : le parcours des occurences dans les deux sens et la possibilité de faire des mises à jour sur une occurence.

Concernant le parcours, il est possible de préciser trois modes de fonctionnement :

- forward−only : parcours sequentiel de chaque occurence (java.sql.ResultSet.TYPE_FORWARD_ONLY)
- scroll−insensitive : les occurences ne reflètent pas les mises à jour qui peuvent intervenir durant le parcours (java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE)
- scroll−sensitive : les occurences reflètent les mises à jour qui peuvent intervenir durant le parcours (java.sql.ResultSet.TYPE_SCROLL_SENSITIVE)

Il est aussi possible de préciser si le ResultSet peut être mise à jour ou non :

- java.sql.ResultSet.CONCUR_READ_ONLY : lecture seule
- java.sql.resultSet.CONCUR_UPDATABLE : mise à jour possible

C'est à la création d'un objet de type Statement qu'il faut préciser ces deux modes. Si ces deux modes ne sont pas précisés, ce sont les caractéristiques de la version 1.0 de JDBC qui sont utilisées (TYPE_FORWARD_ONLY et CONCUR_READ_ONLY).

Exemple (code jdbc 2.0) :

```
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
   ResultSet.CONCUR_READ_ONLY);
 ResultSet resultSet = statement.executeQuery("SELECT nom, prenom FROM employes");
```

Le support de ces fonctionnalités est optionnel pour un pilote. L'objet DatabaseMetadata possède la méthode supportsResultSetType() qui attend en paramètre une constante qui représente une caractéristique : la méthode renvoie un booléen qui indique si la caractéristique est supportée ou non.

A la création du ResultSet, le curseur est positionné avant la première occurence à traiter. Pour se déplacer dans l'ensemble des occurences, il y a toujours la méthode next() pour se déplacer sur le suivant mais aussi plusieurs autres méthodes pour permettre le parcours des occurences en fonctions du mode utilisé dont les principales sont :

Méthode	Rôle
boolean isBeforeFirst()	renvoie un booleen qui indique si la position courante du curseur se trouve avant la première ligne
boolean isAfterLast()	renvoie un booleen qui indique si la position courante du curseur se trouve après la dernière ligne
boolean isFirst()	renvoie un booleen qui indique si le curseur est positionné sur la première ligne
boolean isLast()	renvoie un booleen qui indique si le curseur est positionné sur la dernière ligne
boolean first()	déplace le curseur sur la première ligne
boolean last()	déplace le curseur sur la dernière ligne
boolean absolute()	déplace le curseur sur la ligne dont le numéro est fournie en paramètre à partir du début si il est positif et à partir de la fin si il est négatif. 1 déplace sur la première ligne, −1 sur la dernière, −2 sur l'avant dernière ...
boolean relative(int)	déplace le curseur du nombre de lignes fourni en paramètre par rapport à la position courante du curseur. Le paramètre doit être négatif pour se déplacer vers le début et positif pur se déplacer vers la fin. Avant l'appel de cette méthode, il faut obligatoirement que le curseur soit positionné sur une ligne.
boolean previous()	déplace le curseur sur la ligne précédente. Le booleen indique si la première occurence est dépassée.

void afterLast()	déplace le curseur après la dernière ligne
void beforeFirst()	déplace le curseur avant la première ligne
int getRow()	renvoie le numéro de la ligne courante

Exemple (code jdbc 2.0) :

```
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
   ResultSet.CONCUR_READ_ONLY);
 ResultSet resultSet = statement.executeQuery(
   "SELECT nom, prenom FROM employes ORDER BY nom");
 resultSet.afterLast();
 while (resultSet.previous()) {
   System.out.println(resultSet.getString("nom")+
     " "+resultSet.getString("prenom"));
 }
```

Durant le parcours d'un ResultSet, il est possible d'effectuer des mises à jour sur la ligne courante du curseur. Pour cela, il faut déclarer l'objet ResultSet comme acceptant les mises à jour. Avec les versions précédentes de JDBC, il fallait utiliser la méthode executeUpdate() avec une requête SQL.

Maintenant pour réaliser ces mises à jour, JDBC 2.0 propose de les réaliser via des appels de méthodes plutôt que d'utiliser des requêtes SQL.

Méthode	Rôle
updateXXX(String, XXX)	permet de mettre à jour la colonne dont le nom est fourni en paramètre. Le type Java de cette colonne est XXX
updateXXX(int, XXX)	permet de mettre à jour la colonne dont l'index est fourni en paramètre. Le type Java de cette colonne est XXX
updateRow()	permet d'actualiser les modifications réalisées avec des appels à updateXXX()
boolean rowsUpdated()	indique si la ligne courante a été modifiée
deleteRow()	supprime la ligne courante
rowDeleted()	indique si la ligne courante est supprimée
moveToInsertRow()	permet de créer une nouvelle ligne dans l'ensemble de résultat
inserRow()	permet de valider la création de la ligne

Pour réaliser une mise à jour dans la ligne courante désignée par le curseur, il faut utiliser une des méthodes updateXXX() sur chacun des champs à modifier. Une fois toutes les modifications faites dans une ligne, il faut appeler la méthode updateRow() pour reporter ces modifications dans la base de données car les méthodes updateXXX() ne font des mises à jour que dans le jeu de résultats. Les mises à jour sont perdues si un changement de ligne intervient avant l'appel à la méthode updateRow().

La méthode cancelRowUpdates() permet d'annuler toutes les modifications faites dans la ligne. L'appel à cette méthode doit être effectué avant l'appel à la méthode updateRow().

Pour insérer une nouvelle ligne dans le jeu de résultat, il faut tout d'abord appeler la méthode moveToInsertRow(). Cette méthode déplace le curseur vers un buffer dédié à la création d'une nouvelle ligne. Il faut alimenter chacun des champs nécessaires dans cette nouvelle ligne. Pour valider la création de cette nouvelle ligne, il faut appeler la méthode insertRow().

Pour supprimer la ligne courante, il faut appeler la méthode deleteRow(). Cette méthode agit sur le jeu de résultats et sur la base de données.

22.12.2. Les mises à jour de masse (Batch Updates)

JDBC 2.0 permet de réaliser des mises à jour de masse en regroupant plusieurs traitements pour les envoyer en une seule fois au SGBD. Ceci permet d'améliorer les performances surtout si le nombre de traitements est important.

Cette fonctionnalité n'est pas obligatoirement supportée par le pilote. La méthode supportsBatchUpdate() de la classe DatabaseMetaData permet de savoir si elle est utilisable avec le pilote.

Plusieurs méthodes ont été ajoutées à l'interface Statement pour pouvoir utiliser les mises à jour de masse :

Méthode	Rôle
void addBatch(String)	permet d'ajouter une chaîne contenant une requete SQL
int[] executeBatch()	permet d'exécuter toutes les requêtes. Ell renvoit un tableau d'entier qui contient pour chaque requête, le nombre de mises à jour affectuées.
void clearBatch()	supprime toutes les requêtes stockées

Lors de l'utilisation de batchupdate, il est préférable de positionner l'attribut autocommit à false afin de faciliter la gestion des transactions et le traitement d'une erreur dans l'exécution d'un ou plusieurs traitements.

Exemple (code jdbc 2.0) :

```
connection.setAutoCommit(false);
 Statement statement = connection.createStatement();

 for(int i=0; i<10 ; i++) {
    statement.addBatch("INSERT INTO personne VALUES('nom"+i+"','prenom"+i+"')");
 }
 statement.executeBatch();
```

Une exception particulière peut être levée en plus de l'exception SQLException lors de l'exécution d'une mise à jour de masse. L'exception SQLException est levée si une requête SQL d'intérrogation doit être exécutée (requête de type SELECT). L'exception BatchUpdateException est levée si une des requêtes de mise à jour échoue.

L'exception BatchUpdateException possède une méthode getUpdateCounts() qui renvoie un tableau d'entier qui contient le nombre d'occurences impactées par chaque requête réussie.

22.12.3. Le package javax.sql

Ce package est une extension à l'API JDBC qui propose des fonctionnalités pour les développements coté serveur. C'est pour cette raison, que cette extension est uniquement intégrée à J2EE.

Les principales fonctionnalités proposées sont :

- une nouvelle interface pour assurer la connection : l'interface DataSource
- les pools de connections
- les transactions distribuées
- l'API Rowset

DataSource et Rowset peuvent être utilisées directement. Les pools de connections et les transactions distribuées sont utilisés par une implémentation dans les serveurs d'applications pour fournir ces services.

22.12.4. La classe DataSource

La classe DataSource propose de fournir une meilleure alternative à la classe DriverManager pour assurer la connection à une base de données.

Elle représente une connection physique à une base de données. Les fournisseurs de pilotes proposent une implémentation de l'interface DataSource.

L'utilisation d'un objet DataSource est obligatoire pour pouvoir utiliser un pool de connection et les transactions distribuées. Une fois créé un objet de type DataSource doit être enregistré dans un service de nommage. Il suffit alors d'utiliser JNDI pour obtenir une instance de classe DataSource.

Exemple

```
...
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("jdbc/applicationDB");
Connection con = ds.getConnection("admin", "mpadmin");
...
```

La suite de cette section sera développée dans une version future de ce document

22.12.5. Les pools de connexion

Un pool de connexions permet de maintenir un ensemble de connections établies vers une base de données qui sont réutilisables. L'établissement d'une connexion est très couteux en ressources. L'intérêt du pool de connexions est de limiter le nombre de ces créations et ainsi d'améliorer les performances surtout si le nombre de connexions est important.

La suite de cette section sera développée dans une version future de ce document

22.12.6. Les transactions distribuées

Les connexions obtenues à partir d'un objet DataSource peuvent participer à une transaction distribuée.

La suite de cette section sera développée dans une version future de ce document

22.12.7. L'API RowSet

Cette section sera développée dans une version future de ce document

22.13. JDBC 3.0

La version 3.0 de l'API JDBC a été intégrée au J2SE 1.4.

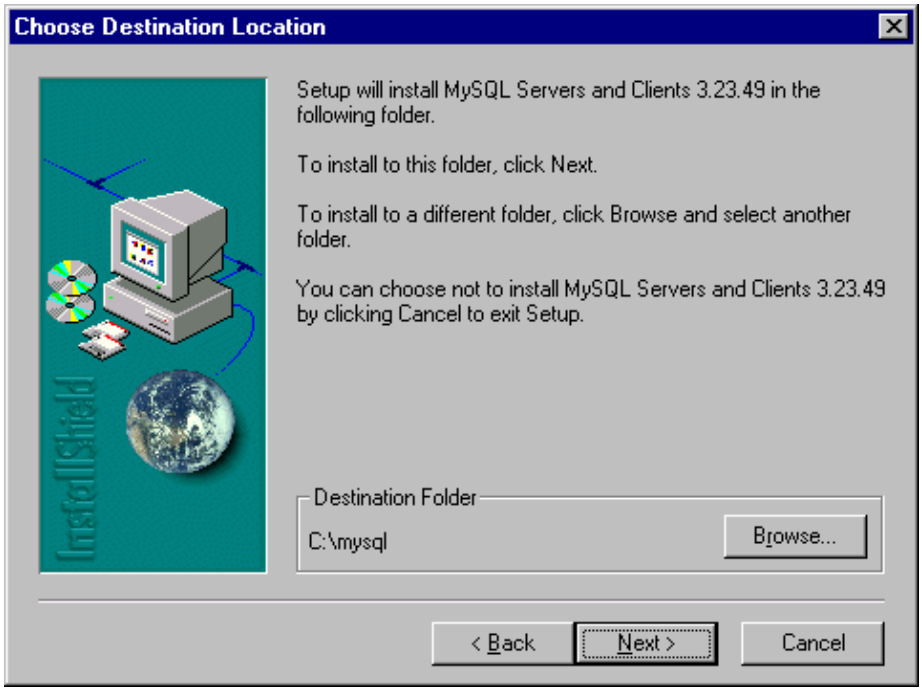
La suite de cette section sera développée dans une version future de ce document

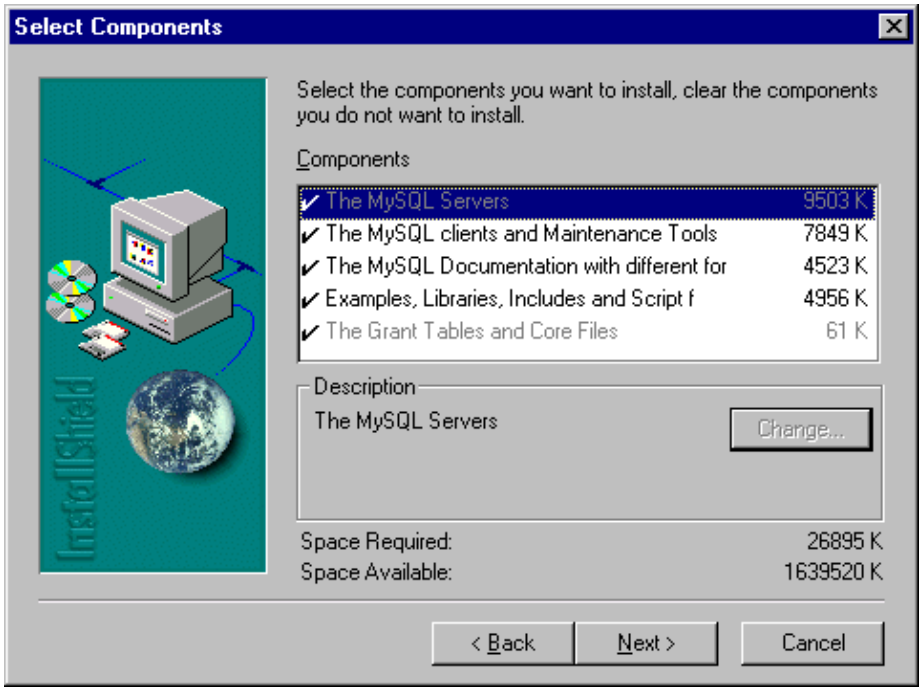
22.14. MySQL et Java

MySQL est une des bases de données open source les plus populaires.

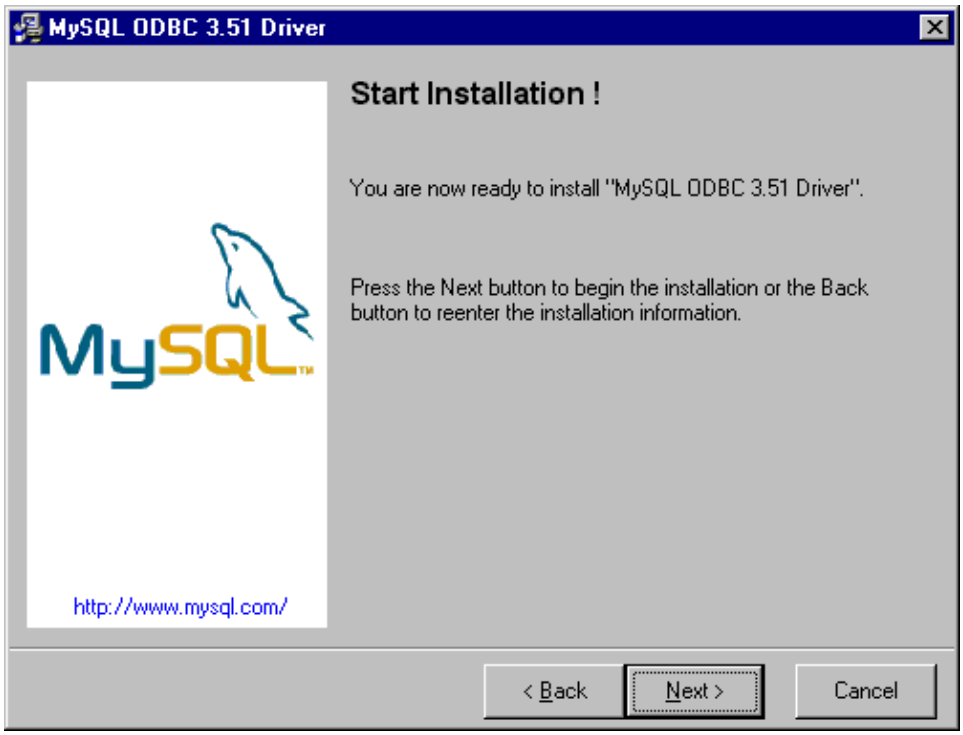
22.14.1. Installation sous Windows

Il suffit de télécharger le fichier mysql−3.23.49−win.zip sur le site www.mysql.com, de décompresser ce fichier dans un repertoire et d'exécuter le fichier setup.exe





Il faut ensuite télécharger le pilote ODBC, MyODBC−3.51.03.exe, et l'exécuter



22.14.2. Opérations de base avec MySQL

Cette section est une présentation rapide de quelques fonctionnalités de base pour pouvoir utiliser MySQL. Pour un complément d'informations sur toutes les possibilités de MySQL, consultez la documentation de cet excellent outil.

Pour utiliser MySQL, il faut s'assurer que le serveur est lancé sinon il faut exécuter la command c:\mysql\bin\mysqld−max

Pour exécuter des commandes SQL, il faut utiliser l'outil c:\mysql\bin\mysql. Cet outil est un interpreteur de commandes en mode console.

Exemple : pour voir les databases existantes

```
mysql>show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)
```

Un des premières choses à faire, c'est de créer une base de données qui va recevoir les différentes tables.

Exemple : Pour créer une nouvelle base de données nommée ‘testjava'

```
mysql> create database testjava;
Query OK, 1 row affected (0.00 sec)

mysql>use testjava;
Database changed
```

Cette nouvelle base de données ne contient aucune table. Il faut créer la ou les tables utiles aux développements.

Exemple : Création d'une table nommée personne contenant trois champs : nom, prenom et date de naissance

```
mysql> show tables;
Empty set (0.06 sec)

mysql> create table personne (nom varchar(30), prenom varchar(30), datenais date
);
Query OK, 0 rows affected (0.00 sec)

mysql>show tables;
+--------------------+
| Tables_in_testjava |
+--------------------+
| personne           |
+--------------------+
1 row in set (0.00 sec)
```

Pour voir la définition d'une table, il faut utiliser la commande DESCRIBE :

Exemple : voir la définition de la table

```
mysql> describe personne;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| nom      | varchar(30) | YES  |     | NULL    |       |
| prenom   | varchar(30) | YES  |     | NULL    |       |
| datenais | date        | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Cette table ne contient aucun enregistrement. Pour ajouter un enregistrement, il faut utiliser la command SQL insert.

Exemple : insertion d'une ligne dans la table

```
mysql> select * from personne;
Empty set (0.00 sec)

mysql> insert into personne values ('Nom 1','Prenom 1','1970-08-11');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> select * from personne;
+-------+----------+------------+
| nom   | prenom   | datenais   |
+-------+----------+------------+
| Nom 1 | Prenom 1 | 1970-08-11 |
+-------+----------+------------+
1 row in set (0.00 sec)
```

Il existe des outils graphiques libres ou commerciaux pour faciliter l'administration et l'utilisation de MySQL.

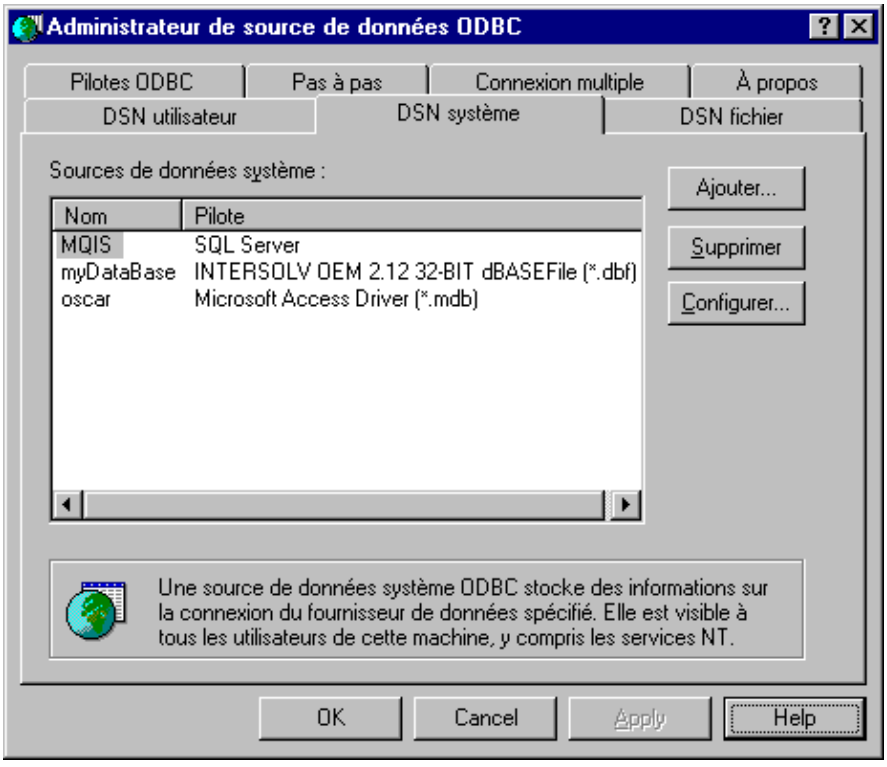
22.14.3. Utilisation de MySQL avec Java via ODBC

Sous Windows, il est possible d'utiliser une base de données MySQL avec Java en utilisant ODBC. Dans ce cas, il faut définir une source de données ODBC sur la base de données et l'utiliser avec le pilote de type 1 fourni en standard avec J2SE.

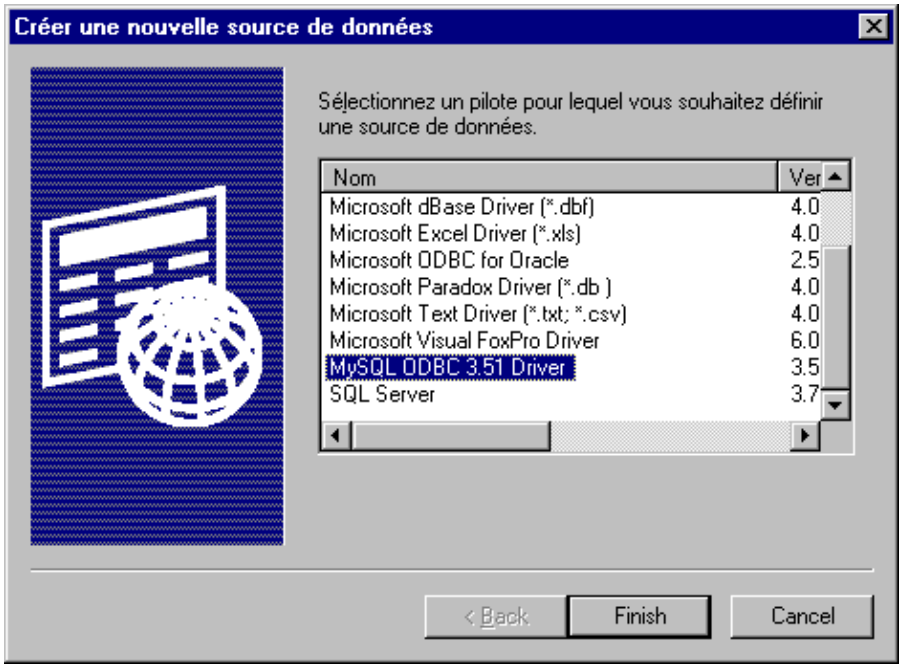
22.14.3.1. Déclaration d'une source de données ODBC vers la base de données

Dans le panneau de configuration, cliquez sur l'icône « Source de données ODBC ».

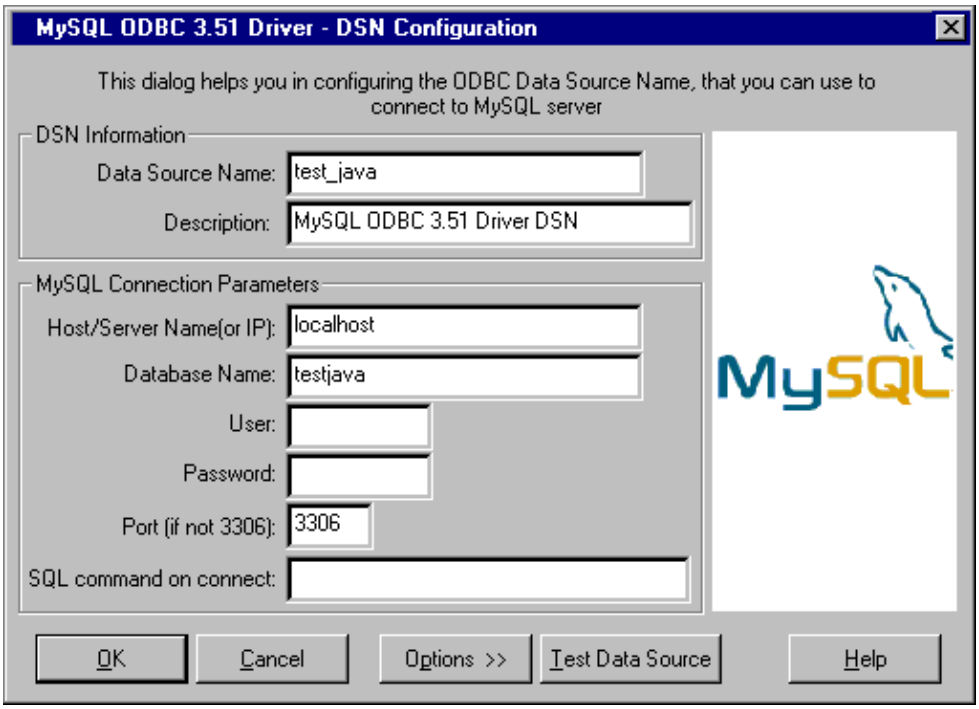
Le plus simple est de créer une source de données Systeme qui pourra être utilisée par tous les utilisateurs en cliquant sur l'onglet " DSN système "



Pour ajouter une nouvelle source de données, il suffit de cliquer sur le bouton "Ajouter ... ". Une boîte de dialogue permet de sélectionner le type de pilote qui sera utilisé par la source de données.

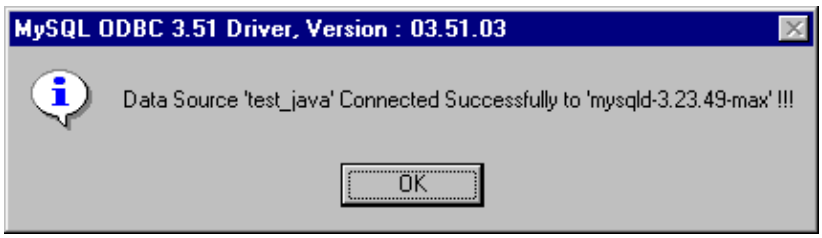


Il faut sélectionner le pilote MySQL et cliquer sur le bouton "Finish".

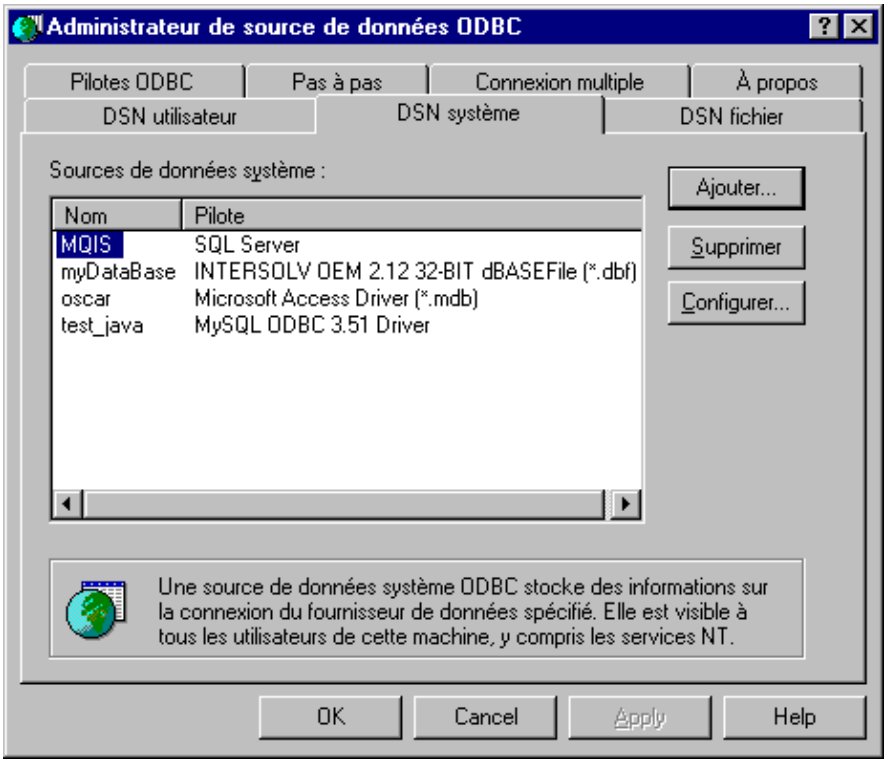


Une nouvelle boîte de dialogue permet de renseigner les informations sur la base de données à utiliser notamment le nom de DSN et le nom de la base de données.

Pour verifier si la connection est possible, il suffit de cliquer sur le bouton « Test Data Source »



Cliquer sur Ok pour fermer la fenêtre et cliquer sur Ok pour valider les paramètres et créer la source de données.



La source de données est créée.

22.14.3.2. Utilisation de la source de données

Pour utiliser la source de données, il faut écrire et tester une classe Java. La seule particularité est l'utilisation du pont JDBC−ODBC comme pilote JDBC et l'URL spécifique à ce pilote qui contient le nom de la source de données définie.

Exemple

```
import java.sql.*;

public class TestJDBC10 {

  private static void affiche(String message) {
    System.out.println(message);
  }

  private static void arret(String message) {
    System.err.println(message);
    System.exit(99);
  }

  public static void main(java.lang.String[] args) {
    Connection con = null;
    ResultSet resultats = null;
    String requete = "";

     // chargement du pilote
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException e) {
      arret("Impossible de charger le pilote jdbc:odbc");
    }

    //connection a la base de données
    affiche("connection a la base de donnees");
    try {

      String DBurl = "jdbc:odbc:test_java";
      con = DriverManager.getConnection(DBurl);
    } catch (SQLException e) {
      arret("Connection à la base de donnees impossible");
    }
```

```
    //creation et execution de la requête
    affiche("creation et execution de la requête");
    requete = "SELECT * FROM personne";

    try {
      Statement stmt = con.createStatement();
      resultats = stmt.executeQuery(requete);
    } catch (SQLException e) {
      arret("Anomalie lors de l'execution de la requête");
    }

    //parcours des données retournees
    affiche("parcours des données retournees");
    try {
      ResultSetMetaData rsmd = resultats.getMetaData();
      int nbCols = rsmd.getColumnCount();
      boolean encore = resultats.next();

      while (encore) {

        for (int i = 1; i <= nbCols; i++)
          System.out.print(resultats.getString(i) + "");

        System.out.println();

        encore = resultats.next();
      }

      resultats.close();
    } catch (SQLException e) {
      arret(e.getMessage());
    }

    affiche("fin du programme");
    System.exit(0);
  }
}
```

Resultat :

```
C:\$user>javac TestJDBC10.java
C:\$user>java TestJDBC10
connection a la base de donnees
creation et execution de la requ_te
parcours des donn_es retournees
Nom 1 Prenom 1 1970-08-11
fin du programme
```

22.14.4. Utilisation de MySQL avec Java via un pilote JDBC

mm.mysql est un pilote JDBC de type IV développé sous licence LGPL par Mark Matthews pour accéder à une base de données MySQL.

Le téléchargement du pilote JDBC se fait sur le site http://mmmysql.sourceforge.net/ . Le fichier mm.mysql-2.0.14-you-must-unjar-me.jar contient les sources et les binaires du pilote.

Pour utiliser cette archive, il faut la décompresser, par exemple dans le répertoire d'installation de mysql.

Il faut s'assurer que les fichiers jar sont accessibles dans le classpath ou les préciser manuellement lors de la compilation et de l'exécution comme dans l'exemple ci dessous.

Exemple

```
import java.sql.*;

public class TestJDBC11 {
```

```
  private static void affiche(String message) {
    System.out.println(message);
  }

  private static void arret(String message) {
    System.err.println(message);
    System.exit(99);
  }

  public static void main(java.lang.String[] args) {
    Connection con = null;
    ResultSetresultats = null;
    String requete = "";

    // chargement du pilote
    try {
      Class.forName("org.gjt.mm.mysql.Driver").newInstance();
    } catch (Exception e) {
      arret("Impossible decharger le pilote jdbc pour mySQL");
    }

    //connection a la base de données
    affiche("connection a la base de donnees");
    try {

      String DBurl = "jdbc:mysql://localhost/testjava";
      con = DriverManager.getConnection(DBurl);
    } catch (SQLException e) {
      arret("Connection a la base de donnees impossible");
    }

    //creation et execution de la requête
    affiche("creation et execution dela requête");
    requete = "SELECT * FROM personne";

    try {
      Statement stmt = con.createStatement();
      resultats = stmt.executeQuery(requete);
    } catch (SQLException e) {
      arret("Anomalie lors de l'execution de la requete");
    }

    //parcours des données retournees
    affiche("Parcours des donnees retournees");
    try {
      ResultSetMetaData rsmd = resultats.getMetaData();
      int nbCols = rsmd.getColumnCount();
      boolean encore = resultats.next();

      while (encore) {

        for (int i = 1; i <= nbCols; i++)
          System.out.print(resultats.getString(i) + "");

        System.out.println();
        encore = resultats.next();
      }

      resultats.close();
    } catch (SQLException e) {
      arret(e.getMessage());
    }

    affiche("fin du programme");
    System.exit(0);
  }
}
```

Le programme est identique au précédent utilisant ODBC sauf :

- le nom de la classe du pilote

- l'URL de connection à la base qui dépend du pilote

Resultat :

```
C:\$user>javac −classpath c:\j2sdk1.4.0−rc\jre\lib\mm.mysql−2.0.14−bin.jar TestJDBC11.java
C:\$user>
C:\$user>java −cp .;c:\j2sdk1.4.0−rc\jre\lib\mm.mysql−2.0.14−bin.jar TestJDBC11
connection a la base de donnees
creation et execution de la requ_te
Parcours des donnees retournees
Nom 1 Prenom 1 1970−08−11
fin du programme
```

23. La gestion dynamique des objets et l'introspection

Depuis la version 1.1 de java, il est possible de créer et de gérer dynamiquement des objets.

L'introspection est un mécanisme qui permet de connaître le contenu d'une classe dynamiquement. Il permet notamment de savoir ce que contient une classe sans en avoir les sources. Ces mécanismes sont largement utilisés dans des outils de type IDE (Integrated Development Environnement : environnement de développement intégré).

Pour illustrer ces différents mécanismes, ce chapitre va construire une classe qui proposera un ensemble de méthodes pour obtenir des informations sur une classe.

Les différentes classes utiles pour l'introspection sont rassemblées dans le package java.lang.reflect.

Voici le début de cette classe qui attend dans son constructeur une chaîne de caractères précisant la classe sur laquelle elle va travailler.

Exemple (code Java 1.1) :

```
import java.util.*;
import java.lang.reflect.*;
public class ClasseInspecteur {
  private Class classe;
  private String nomClasse;
  public ClasseInspecteur(String nomClasse) {
    this.nomClasse = nomClasse;
    try {
      classe = Class.forName(nomClasse);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Ce chapitre contient plusieurs sections :

23.1. La classe Class

Les instances de la classe Class sont des objets représentant les classes du langage. Il y aura une instance représentant chaque classes utilisées : par exemple la classe String, la classe Frame, la classe Class, etc ... . Ces instances sont crées automatiquement par la machine virtuelle lors du chargement de la classe. Il est ainsi possible de connaître les caractéristiques d'une classe de façon dynamique en utilisant les méthodes de la classe Class. Les applications telles que les debuggers, les inspecteurs d'objets et les environnement de développement doivent faire une analyse des objets qu'ils manipulent en utilisant ces mécanismes.

La classe Class est définie dans le package java.lang.

La classe Class permet :

- de décrire une classe ou une interface par introspection : obtenir son nom, sa classe mère, la liste de ces méthodes, de ses variables de classe, de ses constructeurs et variables d'instances, etc ...
- d'agir sur une classe en envoyant, à un objet Class des messages comme à tout autre objet. Par exemple, créer dynamiquement à partir d'un objet Class une nouvelle instance de la classe représentée

23.1.1. Obtenir un objet de la classe Class

La classe Class ne possède pas de constructeur public mais il existe plusieurs façons d'obtenir un objet de la classe Class.

23.1.1.1. Connaître la classe d'un objet

La méthode getClass() définit dans la classe Object renvoie une instance de la classe Class. Par héritage, tout objet java dispose de cette méthode.

Exemple (code Java 1.1) :

```
package introspection;

public class TestGetClass {
   public static void main(java.lang.String[] args) {
      String chaine = "test";
      Class classe = chaine.getClass();
      System.out.println("classe de l'objet chaine = "+classe.getName());
   }
}
```

Résultat :

```
classe de l'objet chaine = java.lang.String
```

23.1.1.2. Obtenir un objet Class à partir d'un nom de classe

La classe Class possède une méthode statique forName() qui permet à partir d'une chaîne de caractères désignant une classe d'instancier un objet de cette classe et de renvoyer un objet de la classe Class pour cette classe.

Cette méthode peut lever l'exception ClassNotFoundException.

Exemple (code Java 1.1) :

```
public class TestForName {
   public static void main(java.lang.String[] args) {
      try {
         Class classe = Class.forName("java.lang.String");
         System.out.println("classe de l'objet chaine = "+classe.getName());
      }
      catch(Exception e) {
         e.printStackTrace();
      }
   }
}
```

Résultat :

```
classe de l'objet chaîne = java.lang.String
```

23.1.1.3. Une troisième façon d'obtenir un objet Class

Il est possible d'avoir un objet de la classe Class en écrivant type.class ou type est le nom d'une classe.

Exemple (code Java 1.1) :

```
package introspection;
public class TestClass {
   public static void main(java.lang.String[] args) {
      Class c = Object.class;
      System.out.println("classe de Object  = "+c.getName());
   }
}
```

Résultat :

```
classe de Object  = java.lang.Object
```

23.1.2. Les méthodes de la classe Class

La classe Class fournie de nombreuses méthodes pour obtenir des informations sur la classe qu'elle représente. Voici les principales méthodes :

Méthodes	Rôle
static Class forName(String)	Instancie un objet de la classe dont le nom est fourni en paramètre et renvoie un objet Class la représentant
Class[] getClasses()	Renvoie les classes et interfaces publiques qui sont membres de la classe
Constructor[] getConstructors()	Renvoie les constructeurs publics de la classe
Class[] getDeclaredClasses()	Renvoie un tableau des classes définies comme membre dans la classe
Constructor[] getDeclaredConstructors()	Renvoie tous les constructeurs de la classe
Field[] getDeclaredFields()	Renvoie un tableau de tous les attributs définis dans la classe
Method getDeclaredMethods()	Renvoie un tableau de toutes les méthodes
Field getFields()	Renvoie un tableau des attributs publics
Class[] getInterfaces()	Renvoie un tableau des interfaces implémentées par la classe
Method getMethod()	Renvoie un tableau des methodes publiques de la classe incluant celles héritées
int getModifiers()	Renvoie un entier qu'il faut décoder pour connaître les modificateurs de la classe
Package getPackage()	Renvoie le package de la classe
Classe getSuperClass()	Renvoie la classe mère de la classe
boolean isArray()	Indique si la classe est un tableau
boolean IsInterface()	Indique si la classe est une interface
Object newInstance()	Permet de créer une nouvelle instance de la classe

23.2. Rechercher des informations sur une classe

En utilisant les méthodes de la classe Class, il est possible d'obtenir quasiment toutes les informations sur une classe.

23.2.1. Rechercher la classe mère d'une classe

La classe Class possède une méthode getSuperClass() qui retourne un objet de la classe Class représentant la classe mère si elle existe sinon elle retourne null.

Pour obtenir toute la hiérarchie d'une classe il suffit d'appeler successivement cette méthode sur l'objet qu'elle a retourné.

Exemple (code Java 1.1) : méthode qui retourne un vecteur contenant les classes mères

```
public Vector getClassesParentes() {

  Vector cp = new Vector();

  Class sousClasse = classe;
  Class superClasse;

  cp.add(sousClasse.getName());
  superClasse = sousClasse.getSuperclass();
  while (superClasse != null) {
    cp.add(0,superClasse.getName());
    sousClasse = superClasse;
    superClasse = sousClasse.getSuperclass();
  }
  return cp;
}
```

23.2.2. Rechercher les modificateurs d'une classe

La classe Class possède une méthode getModifiers() qui retourne un entier représentant les modificateurs de la classe. Pour décoder cette valeur, la classe Modifier possède plusieurs méthodes qui attendent cet entier en paramètre et qui retourne un booléen selon leur fonction : isPublic, isAbstract, isFinal ...

La classe Modifier ne contient que des constantes et des méthodes statiques qui permettent de déterminer les modificateurs d'accès :

Méthode	Rôle
boolean isAbstract(int)	Renvoie true si le paramètre contient le modificateur abstract
boolean isFinal(int)	Renvoie true si le paramètre contient le modificateur final
boolean isInterface(int)	Renvoie true si le paramètre contient le modificateur interface
boolean isNative(int)	Renvoie true si le paramètre contient le modificateur native
boolean isPrivate(int)	Renvoie true si le paramètre contient le modificateur private
boolean isProtected(int)	Renvoie true si le paramètre contient le modificateur protected
boolean isPublic(int)	Renvoie true si le paramètre contient le modificateur public
boolean isStatic(int)	Renvoie true si le paramètre contient le modificateur static
boolean isSynchronized(int)	Renvoie true si le paramètre contient le modificateur synchronized
boolean isTransient(int)	Renvoie true si le paramètre contient le modificateur transient
boolean isVolatile(int)	Renvoie true si le paramètre contient le modificateur volatile

Ces méthodes étant static il est inutile d'instancier un objet de type Modifier pour utiliser ces méthodes.

Exemple (code Java 1.1) :

```
public Vector getModificateurs() {
```

```
    Vector cp = new Vector();
    int m = classe.getModifiers();
    if (Modifier.isPublic(m))
      cp.add("public");
    if (Modifier.isAbstract(m))
      cp.add("abstract");
    if (Modifier.isFinal(m))
      cp.add("final");

    return cp;
  }
```

23.2.3. Rechercher les interfaces implémentées par une classe

La classe Class possède une méthode getInterfaces() qui retourne un tableau d'objet de type Class contenant les interfaces implémentées par la classe.

Exemple (code Java 1.1) :

```
  public Vector getInterfaces() {

    Vector cp = new Vector();

    Class[] interfaces = classe.getInterfaces();
    for (int i = 0; i < interfaces.length; i++) {
      cp.add(interfaces[i].getName());
    }

    return cp;
  }
```

23.2.4. Rechercher les champs publics

La classe Class possède une méthode getFields() qui retourne les attributs public de la classe. Cette méthode retourne un tableau d'objet de type Field.

La classe Class possède aussi une méthode getField() qui attend en paramètre un nom d'attribut et retourne un objet de type Field si celui ci est défini dans la classe ou dans une de ses classes mères. Si la classe ne contient pas d'attribut dont le nom correspond au paramètre fourni, la méthode getField() lève une exception de la classe NoSuchFieldException.

La classe Field représente un attribut d'une classe ou d'une interface et permet d'obtenir des informations cet attribut. Elle possède plusieurs méthodes :

Méthode	Rôle
String getName()	Retourne le nom de l'attribut
Class getType()	Retourne un objet de type Class qui représente le type de l'attribut
Class getDeclaringClass()	Retourne un objet de type Class qui représente la classe qui définie l'attribut
int getModifiers()	Retourne un entier qui décrit les modificateurs d'accès. Pour les connaître précisément il faut utiliser les méthodes static de la classe Modifier.
Object get(Object)	Retourne la valeur de l'attribut pour l'instance de l'objet fourni en parameter. Il existe aussi plusieurs méthodes getXXX() ou XXX représente un type primitf et qui la renvoie la valeur dans ce type.

Exemple (code Java 1.1) :

```
  public Vector getChampsPublics() {

    Vector cp = new Vector();

    Field[] champs = classe.getFields();
```

```
    for (int i = 0; i < champs.length; i++)
      cp.add(champs[i].getType().getName()+" "+champs[i].getName());
    return cp;
  }
```

23.2.5. Rechercher les paramètres d'une méthode ou d'un constructeur

L'exemple ci dessous présente une méthode qui permet de formater sous forme de chaîne de caractères les paramètres d'une méthode fournis sous la forme d'un tableau d'objets de type Class.

Exemple (code Java 1.1) :

```
  private String rechercheParametres(Class[] classes) {

    StringBuffer param = new StringBuffer("(");

    for (int i = 0; i < classes.length; i ++) {

      param.append(formatParametre(classes[i].getName()));
      if (i < classes.length - 1)
        param.append(", ");

    }
    param.append(")");

    return param.toString();

  }
```

La méthode getName() de la classe Class renvoie une chaîne de caractères formatée qui précise le type de la classe. Ce type est représenté par une chaîne de caractères qu'il faut décoder pour obtenir le type.

Si le type de la classe est un tableau alors la chaîne commence par un nombre de caractère '[' correspondant à la dimension du tableau.

Ensuite la chaîne contient un caractère qui précise un type primitif ou un objet. Dans le cas d'un objet, le nom de la classe de l'objet avec son package complet est contenu dans la chaîne suivi d'un caractère ';'.

Caractère	Type
B	byte
C	char
D	double
F	float
I	int
J	long
L*classname;*	*classe ou interface*
S	short
Z	boolean

Exemple :

La méthode getName() de la classe Class représentant un objet de type float[10][5] renvoie « [[F »

Pour simplifier les traitements, la méthode formatParametre() ci dessous retourne une chaîne de caractères qui décode le contenu de la chaîne retournée par la méthode getName() de la classe Class.

Exemple (code Java 1.1) :

```
private String formatParametre(String s) {

  if (s.charAt(0) == '[') {

    StringBuffer param = new StringBuffer("");

    int dimension = 0;
    while (s.charAt(dimension) == '[') dimension++;

    switch(s.charAt(dimension)) {
      case 'B' : param.append("byte");break;
      case 'C' : param.append("char");break;
      case 'D' : param.append("double");break;
      case 'F' : param.append("float");break;
      case 'I' : param.append("int");break;
      case 'J' : param.append("long");break;
      case 'S' : param.append("short");break;
      case 'Z' : param.append("boolean");break;
      case 'L' : param.append(s.substring(dimension+1,s.indexOf(";")));
    }

    for (int i =0; i < dimension; i++)
      param.append("[]");

    return param.toString();
  }
  else return s;

}
```

23.2.6. Rechercher les constructeurs de la classe

La classe Class possède une méthode getConstructors() qui retourne un tableau d'objet de type Constructor contenant les constructeurs de la classe.

La classe Constructor représente un constructeur d'une classe. Elle possède plusieurs méthodes :

Méthode	Rôle
String getName()	Retourne le nom du constructeur
Class[] getExceptionTypes()	Retourne un tableau de type Class qui représente les exceptions qui peuvent être propagées par le constructeur
Class[] getParametersType()	Retourne un tableau de type Class qui représente les paramètres du constructeur
int getModifiers()	Retourne un entier qui décrit les modificateurs d'accès. Pour les connaître précisément il faut utiliser les méthodes static de la classe Modifier.
Object newInstance(Object[])	Instancie un objet en utilisant le constructeur avec les paramètres fournis à la méthode

Exemple (code Java 1.1) :

```
public Vector getConstructeurs() {

  Vector cp = new Vector();
  Constructor[] constructeurs = classe.getConstructors();
  for (int i = 0; i < constructeurs.length; i++) {
    cp.add(rechercheParametres(constructeurs[i].getParameterTypes()));

  }

  return cp;
}
```

L'exemple ci dessus utilise la méthode rechercherParamètres() définie précédemment pour simplifier les traitements.

23.2.7. Rechercher les méthodes publiques

Pour consulter les méthodes d'un objet, il faut obtenir sa classe et lui envoyer le message getMethod(), qui renvoie les méthodes publiques qui sont déclarées dans la classe et qui sont héritées des classes mères.

Elle renvoie un tableau d'instances de la classe Method du package java.lang.reflect.

Une méthode est caractérisée par un nom, une valeur de retour, une liste de paramètres, une liste d'exceptions et une classe d'appartenance.

La classe Method contient plusieurs méthodes :

Méthode	Rôle
Class[] getParameterTypes	Renvoie un tableau de classes représentant les paramètres.
Class getReturnType	Renvoie le type de la valeur de retour de la méthode.
String getName()	Renvoie le nom de la méthode
int getModifiers()	Renvoie un entier qui représentent les modificateur d'accès
Class[] getExceptionTypes	Renvoie un tableau de classes contenant les exceptions propagées par la méthode
Class getDeclaringClass[]	Renvoie la classe qui définit la méthode

Exemple (code Java 1.1) :

```
public Vector getMethodesPubliques() {

  Vector cp = new Vector();
  Method[] methodes = classe.getMethods();
  for (int i = 0; i < methodes.length; i++) {
    StringBuffer methode = new StringBuffer();

    methode.append(formatParametre(methodes[i].getReturnType().getName()));
    methode.append(" ");
    methode.append(methodes[i].getName());
    methode.append(rechercheParametres(methodes[i].getParameterTypes()));

    cp.add(methode.toString());

  }
  return cp;
}
```

L'exemple ci dessus utilise les méthodes formatParametre() et rechercherParamètres() définies précédemment pour simplifier les traitements.

23.2.8. Rechercher toutes les méthodes

Pour consulter toutes les méthodes d'un objet, il faut obtenir sa classe et lui envoyer le message getDeclareMethods(), qui renvoie toutes les méthodes qui sont déclarées dans la classe et qui sont héritées des classes mères quelque soit leur accessibilité.

Elle renvoie un tableau d'instances de la classe Method du package java.lang.reflect.

Exemple (code Java 1.1) :

```
public Vector getMethodes() {

  Vector cp = new Vector();
  Method[] methodes = classe.getDeclareMethods();
  for (int i = 0; i < methodes.length; i++) {
    StringBuffer methode = new StringBuffer();

    methode.append(formatParametre(methodes[i].getReturnType().getName()));
    methode.append(" ");
    methode.append(methodes[i].getName());
    methode.append(rechercheParametres(methodes[i].getParameterTypes()));

    cp.add(methode.toString());

  }

  return cp;
}
```

L'exemple ci dessus utilise les méthodes formatParametre() et rechercherParamètres() définies précédemment pour simplifier les traitements.

23.3. Définir dynamiquement des objets

23.3.1. Définir des objets grâce à la classe Class

Cette section sera développée dans une version future de ce document

23.3.2. Exécuter dynamiquement une méthode

Cette section sera développée dans une version future de ce document

24. L'appel de méthodes distantes : RMI

RMI (Remote Method Invocation) est une technologie développée et fournie par Sun à partir du JDK 1.1 pour permettre de mettre en oeuvre facilement des objets distribués.

Ce chapitre contient plusieurs sections :

24.1. Présentation et architecture de RMI

Le but de RMI est de permettre l'appel, l'exécution et le renvoi du résultat d'une méthode exécutée dans une machine virtuelle différente de celle de l'objet l'appelant. Cette machine virtuelle peut être sur une machine différente pourvu qu'elle soit accessible par le réseau.

La machine sur laquelle s'exécute la méthode distante est appelée serveur.

L'appel coté client d'une telle méthode est un peu plus compliqué que l'appel d'une méthode d'un objet local mais il reste simple. Il consiste à obtenir une référence sur l'objet distant puis à simplement appeler la méthode à partir de cette référence.

La technologie RMI se charge de rendre transparente la localisation de l'objet distant, son appel et le renvoi du résultat.

En fait, elle utilise deux classes particulières, le stub et le skeleton, qui doivent être générées avec l'outil rmic fourni avec le JDK.

Le stub est une classe qui se situe côté client et le skeleton est son homologue coté serveur. Ces deux classes se chargent d'assurer tous les mécanismes d'appel, de communication, d'exécution, de renvoi et de réception du résultat.

24.2. Les différentes étapes pour créer un objet distant et l'appeler avec RMI

Le développement coté serveur se compose de :

- La définition d'une interface qui contient les méthodes qui peuvent être appelées à distance
- L'écriture d'une classe qui implémente cette interface
- L'écriture d'une classe qui instanciera l'objet et l'enregistrera en lui affectant un nom dans le registre de nom RMI (RMI Registry)

Le développement côté client se compose de :

- L'obtention d'une référence sur l'objet distant à partir de son nom
- L'appel à la méthode à partir de cette référence

Enfin, il faut générer les classes stub et skeleton en exécutant le programme rmic avec le fichier source de l'objet distant

24.3. Le développement coté serveur

24.3.1. La définition d'une interface qui contient les méthodes de l'objet distant

L'interface à définir doit hériter de l'interface java.rmi.Remote. Cette interface ne contient aucune méthode mais indique simplement que l'interface peut être appelée à distance.

L'interface doit contenir toutes les méthodes qui seront succeptibles d'être appelées à distance.

La communication entre le client et le serveur lors de l'invocation de la méthode distante peut échouer pour diverses raisons tel qu'un crash du serveur, une rupture de la liaison, etc ...

Ainsi chaque méthode appelée à distance doit déclarer qu'elle est en mesure de lever l'exception java.rmi.RemoteException.

Exemple (code Java 1.1) :

```
package test_rmi;

import java.rmi.*;

public interface Information extends Remote {

   public String getInformation() throws RemoteException;

}
```

24.3.2. L'écriture d'une classe qui implémente cette interface

Cette classe correspond à l'objet distant. Elle doit donc implémenter l'interface définie et contenir le code nécessaire.

Cette classe doit obligatoirement hériter de la classe UnicastRemoteObject qui contient les differents traitements élémentaires pour un objet distant dont l'appel par le stub du client est unique. Le stub ne peut obtenir qu'une seule référence sur un objet distant héritant de UnicastRemoteObject. On peut supposer qu'une future version de RMI sera capable de faire du MultiCast, permettant à RMI de choisir parmi plusieurs objets distants identiques la référence à fournir au client.

La hiérarchie de la classe UnicastRemoteObject est :

java.lang.Object

java.rmi.Server.RemoteObject

java.rmi.Server.RemoteServer

java.rmi.Server.UnicastRemoteObject

Comme indiqué dans l'interface, toutes les méthodes distantes doivent indiquer qu'elles peuvent lever l'exception RemoteException mais aussi le constructeur de la classe. Ainsi, même si le constructeur ne contient pas de code il doit être redéfini pour inhiber la génération du constructeur par défaut qui ne lève pas cette exception.

Exemple (code Java 1.1) :

```
package test_rmi;

import java.rmi.*;

import java.rmi.server.*;

public class TestRMIServer extends UnicastRemoteObject implements Information {

   protected TestRMIServer() throws RemoteException {
      super();
   }

   public String getInformation()throws RemoteException {
      return "bonjour";
   }

}
```

24.3.3. L'écriture d'une classe pour instancier l'objet et l'enregistrer dans le registre

Ces opérations peuvent être effectuées dans la méthode main d'une classe dédiée ou dans la méthode main de la classe de l'objet distant. L'intérêt d'une classe dédiée et qu'elle permet de regrouper toutes ces opérations pour un ensemble d'objets distants.

La marche à suivre contient trois étapes :

- la mise en place d'un security manager dédié qui est facultative
- l'instanciation d'un objet de la classe distante
- l'enregistrement de la classe dans le registre de nom RMI en lui donnant un nom

24.3.3.1. La mise en place d'un security manager

Cette opération n'est pas obligatoire mais elle est recommandée en particulier si le serveur doit charger des classes qui ne sont pas sur le serveur. Sans security manager, il faut obligatoirement mettre à la disposition du serveur toutes les classes dont il aura besoin (Elles doivent être dans le CLASSPATH du serveur). Avec un security manager, le serveur peut charger dynamiquement certaines classes.

Cependant, le chargement dynamique de ces classes peut poser des problèmes de sécurité car le serveur va exécuter du code d'une autre machine. Cet aspect peut conduire à ne pas utiliser de sécurity manager.

Exemple (code Java 1.1) :

```
public static void main(String[] args) {
   try {
      System.out.println("Mise en place du Security Manager ...");
      System.setSecurityManager(new java.rmi.RMISecurityManager());
   } catch (Exception e) {
      System.out.println("Exception capturée: " + e.getMessage());
   }
}
```

24.3.3.2. L'instanciation d'un objet de la classe distante

Cette opération est très simple puisqu'elle consiste simplement en la création d'un objet de la classe de l'objet distant

Exemple (code Java 1.1) :

```
public static void main(String[] args) {

   try {
      System.out.println("Mise en place du Security Manager ...");
      System.setSecurityManager(new java.rmi.RMISecurityManager());
```

```
      TestRMIServer testRMIServer = new TestRMIServer();

   } catch (Exception e) {

      System.out.println("Exception capturée: " + e.getMessage());

   }

}
```

24.3.3.3. L'enregistrement dans le registre de nom RMI en lui donnant un nom

La dernière opération consiste à enregistrer l'objet créé dans le registre de nom en lui affectant un nom. Ce nom est fourni au registre sous forme d'une URL constitué du préfix rmi://, du nom du seveur (hostname) et du nom associé à l'objet précédé d'un slash.

Le nom du serveur peut être fourni « en dur » sous forme d'une constante chaîne de caractères ou peut être dynamiquement obtenu en utilisant la classe InetAddress pour une utilisation en locale.

C'est ce nom qui sera utilisé dans une URL par le client pour obtenir une référence sur l'objet distant.

L'enregistrement se fait en utilisant la méthode rebind de la classe Naming. Elle attend en paramètre l'URL du nom de l'objet et l'objet lui même.

Exemple (code Java 1.1) :

```
public static void main(String[] args) {

   try {

      System.out.println("Mise en place du Security Manager ...");
      System.setSecurityManager(new java.rmi.RMISecurityManager());

      TestRMIServer testRMIServer = new TestRMIServer();

      System.out.println("Enregistrement du serveur");

      Naming.rebind("rmi://"+java.net.InetAddress.getLocalHost()+
         "/TestRMI",testRMIServer);

      // Naming.rebind(";rmi://localhost/TestRMI", testRMIServer);

      System.out.println("Serveur lancé");

   } catch (Exception e) {
      System.out.println("Exception capturée: " + e.getMessage());
   }

}
```

24.3.3.4. Lancement dynamique du registre de nom RMI

Sur le serveur, le registre de nom RMI doit s'exécuter avant de pouvoir enregistrer un objet ou obtenir une référence.

Ce registre peut être lancé en tant qu'application fournie par sun dans le JDK (rmiregistry) comme indiqué dans un chapitre suivant ou être lancé dynamiquement dans la classe qui enregistre l'objet. Ce lancement ne doit avoir lieu qu'une seule et unique fois. Il peut être intéressant d'utiliser ce code si l'on crée une classe dédiée à l'enregistrement des objets distants.

Le code pour exécuter le registre est la méthode createRegistry de la classe java.rmi.registry.LocateRegistry. Cette méthode attend en paramètre un numéro de port.

Exemple (code Java 1.1) :

```
public static void main(String[] args) {

   try {

      java.rmi.registry.LocateRegistry.createRegistry(1099);

      System.out.println("Mise en place du Security Manager ...");
      System.setSecurityManager(new java.rmi.RMISecurityManager());

      ...

   }
}
```

24.4. Le développement coté client

L'appel d'une méthode distante peut se faire dans une application ou dans une applet.

24.4.1. La mise en place d'un security manager

Comme pour le coté serveur, cette opération est facultative.

Le choix de la mise en place d'un sécurity manager côté client suit des règles identiques à celui du côté serveur. Sans son utilisation, il est nécessaire de mettre dans le CLASSPATH du client toutes les classes nécessaires dont la classe stub.

Exemple (code Java 1.1) :

```
public static void main(String[] args) {

   System.setSecurityManager(new RMISecurityManager());

}
```

24.4.2. L'obtention d'une référence sur l'objet distant à partir de son nom

Pour obtenir une référence sur l'objet distant à partir de son nom, il faut utiliser la méthode statique lookup() de la classe Naming.

Cette méthode attend en paramètre une URL indiquant le nom qui référence l'objet distant. Cette URL est composée de plusieurs éléments : le préfix rmi://, le nom du serveur (hostname) et le nom de l'objet tel qu'il a été enregistré dans le registre précédé d'un slash.

Il est préférable de prévoir le nom du serveur sous forme de paramètres de l'application ou de l'applet pour plus de souplesse.

La méthode lookup() va rechercher dans le registre du serveur l'objet et retourner un objet stub. L'objet retourné est de la classe Remote (cette classe est la classe mère de tous les objets distants).

Si le nom fourni dans l'URL n'est pas référencé dans le registre, la méthode lève l'exception NotBoundException.

Exemple (code Java 1.1) :

```
public static void main(String[] args) {

   System.setSecurityManager(new RMISecurityManager());

   try {

      Remote r = Naming.lookup("rmi://vaio/127.0.0.1/TestRMI");

   } catch (Exception e) {
```

```
   }

}
```

24.4.3. L'appel à la méthode à partir de la référence sur l'objet distant

L'objet retourné étant de type Remote, il faut réaliser un cast vers l'interface qui définit les méthodes de l'objet distant. Pour plus de sécurité, on vérifie que l'objet retourné est bien une instance de cette interface.

Un fois le cast réalisé, il suffit simplement d'appeler la méthode.

Exemple (code Java 1.1) :

```
public static void main(String[] args) {

   System.setSecurityManager(new RMISecurityManager());

   try {

      Remote r = Naming.lookup("rmi://vaio/127.0.0.1/TestRMI");

      if (r instanceof Information) {
         String s = ((Information) r).getInformation();
         System.out.println("chaine renvoyée = " + s);
      }

   } catch (Exception e) {
   }
}
```

24.4.4. L'appel d'une méthode distante dans une applet

L'appel d'une méthode distante est la même dans une application et dans une applet.

Seule la mise en place d'un security manager dédié dans les applets est inutile car elles utilisent déjà un sécurity manager (AppletSecurityManager) qui autorise le chargement de classes distantes.

Exemple (code Java 1.1) :

```
package test_rmi;

import java.applet.*;
import java.awt.*;
import java.rmi.*;

public class AppletTestRMI extends Applet {

   private String s;

   public void init() {

      try {
         Remote r = Naming.lookup("rmi://vaio/127.0.0.1/TestRMI");

         if (r instanceof Information) {
            s = ((Information) r).getInformation();
         }
      } catch (Exception e) {
      }

   }

   public void paint(Graphics g) {
      super.paint(g);
      g.drawString("chaine retournée = "+s,20,20);
   }
```

```
}
```

24.5. La génération des classes stub et skeleton

Pour générer ces classes, il suffit d'utiliser l'outil rmic fourni avec le JDK en lui donnant en paramètre le nom de la classe.

Attention la classe doit avoir été compilée : rmic à besoin du fichier .class.

Exemple (code Java 1.1) :

```
rmic test_rmi.TestRMIServer
```

rmic va générer et compiler les classes stub et skeleton respectivement sous le nom TestRMIServer_Stub.class et TestRMIServer_Skel.class

24.6. La mise en oeuvre des objets RMI

La mise en oeuvre et l'utilisation d'objet distant avec RMI nécessite plusieurs étapes :

1. Démarrer le registre RMI sur le serveur soit en utilisant le programme rmiregistry livré avec le JDK soit en exécutant une classe qui éffectue le lancement.
2. exécuter la classe qui instancie l'objet distant et l'enregistre dans le serveur de nom RMI
3. Lancer l'application ou l'applet pour tester.

24.6.1. Le lancement du registre RMI

La commande rmiregistry est fournie avec le JDK.

Il faut la lancer en tache de fond :

Sous Unix : rmiregistry&

Sous Windows : start rmiregistry

Ce registre permet de faire correspondre un objet à un nom et inversement. C'est lui qui est sollicité lors d'un appel aux méthodes Naming.bind() et Naming.lookup()

24.6.2. L'instanciation et l'enregistrement de l'objet distant

Il faut exécuter la classe qui va instancier l'objet distant et l'enregistrer sous son nom dans le registre précédemment lancé.

Pour ne pas avoir de problème, il faut s'assurer que toutes les classes utiles (la classe de l'objet distant, l'interface qui définit les méthodes, le skeleton) sont présentes dans un répertoire défini dans la variable CLASSPATH.

24.6.3. Le lancement de l'application cliente

La suite de cette section sera développée dans une version future de ce document

25. L'internationalisation

La localisation consiste à adapter un logiciel pour s'adapter aux caractéristiques locales de l'environnement d'execution telles que la langue. Le plus gros du travail consiste à traduire toutes les phrases et les mots. Les classes nécessaires sont incluses dans le packetage java.util.

Ce chapitre contient plusieurs sections :

- Les objets de type Locale
- La classe ResourceBundle
- Chemins guidés pour réaliser la localisation : cette section propose plusieurs solutions pour réalisation l'internationalisation.

25.1. Les objets de type Locale

Un objet de type Locale identifie une langue et un pays donné.

25.1.1. Création d'un objet Locale

Exemple (code Java 1.1) :

```
locale_US = new
Locale("en","US") ;

locale_FR = new Locale("fr","FR");
```

Le premier paramètre est le code langue (deux caractères minuscules conformes à la norme ISO−639 : exemple "de" pour l'allemand, "en" pour l'anglais, "fr" pour le français, etc ...)

Le deuxième paramètre est le code pays (deux caractères majuscules conformes à la norme ISO−3166 : exemple : "DE" pour l'Allemagne, "FR" pour la France,"US" pour les Etats Unis, etc ...). Ce paramètre est obligatoire : si le pays n'a pas besoin d'être précisé, il faut fournir une chaîne vide.

Exemple (code Java 1.1) :

```
Locale locale = new Locale("fr", "");
```

Un troisième paramètre peut permettre de préciser d'avantage la localisation par exemple la plateforme d'execution (il ne respecte aucun standard car il ne sera défini que dans l'application qui l'utilise) :

Exemple (code Java 1.1) :

```
Locale locale_unix = new Locale("fr","FR", "UNIX");
Locale locale_windows = new Locale("fr","FR", "WINDOWS");
```

Ce troisième paramètre est optionnel.

La classe Locale définit des constantes pour certaines langues et pays :

Exemple (code Java 1.1) : ces deux lignes sont equivalentes

```
Locale locale_1 = Locale.JAPAN;
Locale locale_2 = new Locale("ja", "JP");
```

Lorsque l'on précise une constante représentant une langue alors le code pays n'est pas défini.

Exemple (code Java 1.1) : ces deux lignes sont equivalentes

```
Locale locale_3 = Locale.JAPANESE;
Locale locale_2 = new Locale("ja", "");
```

Il est possible de rendre la création d'un objet Locale dynamique :

Exemple (code Java 1.1) :

```
static public void main(String[] args) {
   String langue = new String(args[0]);
   String pays = new String(args[1]);
   locale = new Locale(langue, pays);
}
```

Cet objet ne sert que d'identifiant qu'il faut passer à des objets de type ResourceBundle par exemple qui eux possèdent le nécessaire pour réaliser la localisation. En fait, le création d'un objet Locale pour un pays donné ne signifie pas que l'on va pouvoir l'utiliser.

25.1.2. Obtenir la liste des Locales disponibles

La méthode getAvailableLocales() permet de connaître la liste des Locales reconnues par une classe sensible à l'internationalisation

Exemple (code Java 1.1) : avec la classe DateFormat

```
import java.util.*;
import java.text.*;

public class Available {
   static public void main(String[] args) {
      Locale liste[] =
      DateFormat.getAvailableLocales();
      for (int i = 0; i < liste.length; i++)
      {
         System.out.println(liste[i].toString());
         // toString retourne le code langue et le code pays séparé d'un souligné
      }
   }
}
```

La méthode Locale.getDisplayName() peut être utilisée à la place de toString pour obtenir le nom du code langue et du code pays.

25.1.3. L'utilisation d'un objet Locale

Il n'est pas obligatoire de se servir du même objet Locale avec les classes sensibles à l'internationnalisation.

Cependant la plupart des applications utilisent l'objet Locale par défaut initialisé par la machine virtuelle avec les paramètres de la machine hote. La méthode Locale.getDefault() permet de connaître l'objet Locale par défaut.

25.2. La classe ResourceBundle

Il est préférable de définir un ResourceBundle pour chaque catégorie d'objet (exemple un par fenêtre) : ceci rend le code plus clair et plus facile à maintenir, évite d'avoir des ResourceBundle trop importants et réduit l'espace mémoire utilisé car chaque ressource n'est chargée que lorsque l'on en a besoin.

25.2.1. La création d'un objet ResourceBundle

Conceptuellement, chaque ResourceBundle est un ensemble de sous classes qui partage la même racine de nom.

Exemple (code Java 1.1) :

```
TitreBouton
TitreBouton_de
TitreBouton_en_GB
TitreBouton_fr_FR_UNIX
```

Pour sélectionner le ResourceBundle approprié il faut utiliser la méthode ResourceBundle.getBundle().

Exemple (code Java 1.1) :

```
Locale locale = new Locale("fr", "FR");
ResourceBundle messages = ResourceBundle.getBundle("TitreBouton", locale);
```

Le premier argument contient le type d'objet à utiliser (la racine du nom de cet objet).

Le second argument de type Locale permet de déterminer quel fichier sera utilisé : il ajoute le code pays et le code langue séparé par un souligné à la racine du nom.

Si la classe désignée par l'objet Locale n'existe par, alors getBundle recherche celle qui se rapproche le plus. L'ordre de recherche sera le suivant :

Exemple (code Java 1.1) :

```
TitreBouton_fr_CA_UNIX
TitreBouton_fr_FR
TitreBouton_fr
TitreBouton_en_US
TitreBouton_en
TitreBouton
```

Si aucune n'est trouvée alors getBundle lève une exception de type MissingResourceException.

25.2.2. Les sous classes de ResourceBundle

La classe abstraite ResourceBundle possède deux sous classes : PropertyResourceBundle et ListResourceBundle.

La classe ResourceBundle est une classe fléxible : le changement de l'utilisation d'un PropertyResourceBundle en ListResourceBundle se fait sans impact sur le code. La méthode getBundle() recherche le ResourceBundle désiré qu'il soit dans un fichier .class ou propriétés

25.2.2.1. L'utilisation de PropertyResourceBundle

Un PropertyResourceBundle est rattaché à un fichier propriétés. Ces fichiers propriétés ne font pas partie du code source java. Ils ne peuvent contenir que des chaines de caractères. Pour stocker d'autres objets, il faut utiliser des objets ListResourceBundle.

La création d'un fichier propriétés est simple : c'est un fichier texte qui contient des paires clé−valeur. La clé et la valeur sont séparées par un signe =. Chaque paire doit être sur une ligne séparée.

Exemple (code Java 1.1) :

```
texte_suivant = suivant
texte_precedent = precedent
```

Le nom du fichier propriétés par défaut se compose de la racine du nom suivi de l'extension .properties.

Exemple : TitreBouton.properties.

Dans une autre langue, anglais par exemple, le fichier s'appelerait : TitreBouton_en.properties

Exemple (code Java 1.1) :

```
texte_suivant = next
texte_precedent = previous
```

Les clés sont les mêmes, seule la traduction change.

Le nom de fichier TitreBouton_fr_FR.properties contient la racine (Titrebouton), le code langue (fr) et le code pays (FR).

25.2.2.2. L'utilisation de ListResourceBundle

La classe ListResourceBundle gère les ressources sous forme de liste encapsulée dans un objet. Chaque ListResourceBundle est donc rattaché à un fichier .class. On peut y stocker n'importe quel objet spécifique à la localisation.

Les objets ListResourceBundle contiennent des paires clé−valeur. La clé doit être une chaîne qui caractérise l'objet. La valeur est un objet de n'importe quelle classe.

Exemple (code Java 1.1) :

```
class TitreBouton_fr extends ListResourceBundle {
   public Object[][] getContents() {
      return contents;
   }
   static final Object[][] contents = {
      {"texte_suivant", "Suivant"},
      {"texte_precedent", "Precedent"},
   };
}
```

25.2.3. Obtenir un texte d'un objet ResourceBundle

La méthode getString() retourne la valeur de la clé précisée en paramètre.

Exemple (code Java 1.1) :

```
String message_1 = messages.getString("texte_suivant");
String message_2 = TitreBouton.getString("texte_suivant");
```

25.3. Chemins guidés pour réaliser la localisation

25.3.1. L'utilisation d'un ResourceBundle avec un fichier propriétés

Il faut toujours créer le fichier propriété par défaut. Le nom de ce fichier commence avec le nom de base du ResourceBundle et se termine avec le suffix .properties

Exemple (code Java 1.1) :

```
#Exemple de fichier propriété par défaut (TitreBouton.properties)
texte1 = suivant
texte2 = precedent
texte3 = quitter
```

Les lignes de commentaires commencent par un #. Les autres lignes contiennent les paires clé−valeur. Une fois le fichier défini, il ne faut plus modifier la valeur de la clé qui pourrait être appelée dans un programme.

Pour ajouter le support d'autre langue, il faut créer des fichiers propriétés supplémentaires qui contiendront les traductions.
Le fichier est le même, seul le texte contenu dans la valeur change (la valeur de la clé doit être identique).

Exemple (code Java 1.1) :

```
#Exemple de fichier propriété en anglais (TitreBouton_en.properties)
texte1 = next
texte2 = previous
texte3 = quit
```

Lors de la programmation, il faut créer un objet Locale. Il est possible de créer un tableau qui contient la liste des Locale disponibles en fonction des fichiers propriétés créés.

Exemple (code Java 1.1) :

```
Locale[] locales = { Locale.GERMAN, Locale.ENGLISH };
```

Dans cet exemple, l'objet Locale.ENGLISH correspond au fichier TitreBouton_en.properties. L'objet Locale.GERMAN ne possédant pas de fichier propriétés défini, le fichier par défaut sera utilisé.

Il faut créer l'objet ResourceBundle en invoquant la méthode getBundle de l'objet Locale.

Exemple (code Java 1.1) :

```
ResourceBundle titres =ResourceBundle.getBundle("TitreBouton", locales[1]);
```

La méthode getBundle() recherche en premier une classe qui correspond au nom de base, si elle n'existe pas alors elle recherche un fichier de propriétés. Lorsque le fichier est trouvé, elle retourne un objet PropertyResourceBundle qui contient les paires clé−valeur du fichier

Pour retrouver la traduction d'un texte, il faut utiliser la méthode getString() d'un objet ResourceBundle

Exemple (code Java 1.1) :

```
String valeur = titres.getString(key);
```

Lors du deboggage, il peut être utile d'obtenir la liste de paire d'un objet ResourceBundle. La méthode getKeys() retourne un objet Enumeration qui contient toutes les clés de l'objet.

Exemple (code Java 1.1) :

```
ResourceBundle titres =ResourceBundle.getBundle("TitreBouton", locales[1});
Enumeration cles = titres.getKeys();
while (bundleKeys.hasMoreElements()) {
   String cle = (String)cles.nextElement();
   String valeur = titres.getString(cle);
   System.out.println("cle = " + valeur +
      ", " +   "valeur = " + valeur);
}
```

25.3.2. Exemples de classes utilisant PropertiesResourceBundle

Exemple (code Java 1.1) : Sources de la classe I18nProperties

```
/*
Test d'utilisation de la classe PropertiesResourceBundle
pour internationaliser une application
13/02/99
*/

import java.util.*;

/**
 * Description de la classe I18nProperties
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 */
public class I18nProperties {
  /**
   * Constructeur de la classe
   */
  public I18nProperties() {

    String texte;
    Locale locale;
    ResourceBundle res;

    System.out.println("Locale par defaut : ");
    locale = Locale.getDefault();
    res = ResourceBundle.getBundle("I18nPropertiesRessources", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);

    System.out.println("\nLocale anglaise : ");
    locale = new Locale("en","");
    res = ResourceBundle.getBundle("I18nPropertiesRessources", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);
```

```
    System.out.println("\nLocale allemande : "+
       "non définie donc utilisation locale par defaut ");
    locale = Locale.GERMAN;
    res = ResourceBundle.getBundle("I18nPropertiesRessources", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);
  }

 /**
   * Pour tester la classe
   *
   * @param      args[]         arguments passes au programme
   */
  public static void main(String[] args) {
    I18nProperties i18nProperties = new I18nProperties();
  }

}
```

Exemple (code Java 1.1) : Contenu du fichier I18nPropertiesRessources.properties

```
texte_suivant=suivant
texte_precedent=Precedent
```

Exemple (code Java 1.1) : Contenu du fichier I18nPropertiesRessources_en.properties

```
texte_suivant=next
texte_precedent=previous
```

Exemple (code Java 1.1) : Contenu du fichier I18nPropertiesRessources_en_US.properties

```
texte_suivant=next
texte_precedent=previous
```

25.3.3. L'utilisation de la classe ListResourceBundle

Il faut créer autant de sous classes de ListResourceBundle que de langues désirées : ceci va générer un fichier .class pour chacune des langues .

Exemple (code Java 1.1) :

```
TitreBouton_fr_FR.class
TitreBouton_en_EN.class
```

Le nom de la classe doit contenir le nom de base plus le code langue et le code pays séparés par un souligné. A l'intérieur de la classe, un tableau à deux dimensions est initialisé avec les paires clé−valeur. Les clés sont des chaines qui doivent être identiques dans toutes les classes des différentes langues. Les valeurs peuvent être des objets de n'importe quel type.

Exemple (code Java 1.1) :

```
import java.util.*;

public class TitreBouton_fr_FR extends ListResourceBundle {
   public Object[][] getContents() {
      return contents;
   }

   private Object[][] contents = {
      { "texte_suivant", new String(" suivant ")},
      { "Numero", new Integer(4) }
   };
}
```

Il faut définir un objet de type Locale

Il faut créer un objet de type ResourceBundle en appelant la méthode getBundle() de la classe Locale

Exemple (code Java 1.1) :

```
ResourceBundle titres=ResourceBundle.getBundle("TitreBouton", locale);
```

La méthode getBundle() recherche une classe qui commence par TitreBouton et qui est suivie par le code langue et le code pays précisé dans l'objet Locale passé en paramètre

La méthode getObject permet d'obtenir la valeur de la clé passée en paramètres. Dans ce cas une conversion est nécessaire.

Exemple (code Java 1.1) :

```
Integer valeur = (Integer)titres.getObject("Numero");
```

25.3.4. Exemples de classes utilisant ListResourceBundle

Exemple (code Java 1.1) : Sources de la classe I18nList

```
/*
Test d'utilisation de la classe ListResourceBundle
pour internationaliser une application
13/02/99
*/

import java.util.*;

/**
 * Description de la classe I18nList
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 */
public class I18nList {
  /**
   * Constructeur de la classe
   */
  public I18nList() {

    String texte;
    Locale locale;
    ResourceBundle res;

    System.out.println("Locale par defaut : ");
    locale = Locale.getDefault();
    res = ResourceBundle.getBundle("I18nListRessources", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);

    System.out.println("\nLocale anglaise : ");
    locale = new Locale("en","");
    res = ResourceBundle.getBundle("I18nListRessources", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);

    System.out.println("\nLocale allemande : "+
       "non définie donc utilisation locale par defaut ");
    locale = Locale.GERMAN;
```

```
    res = ResourceBundle.getBundle("I18nListRessources", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);
  }

 /**
   * Pour tester la classe
   *
   * @param      args[]         arguments passes au programme
   */
  public static void main(String[] args) {
    I18nList i18nList = new I18nList();
  }

}
```

Exemple (code Java 1.1) : Sources de la classe I18nListRessources

```
/*
test d'utilisation de la classe ListResourceBundle pour
internationaliser une application
13/02/99
*/

import java.util.*;

/**
 * Ressource contenant les traductions françaises
 * langue par defaut de l'application
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 *
 */

public class I18nListRessources extends ListResourceBundle {
  public Object[][] getContents() {
    return contents;
  }

   //tableau des mots clés et des valeurs

   static final Object[][] contents = {
   {"texte_suivant", "Suivant"},
   {"texte_precedent", "Precedent"},
  };
}
```

Exemple (code Java 1.1) : Sources de la classe I18nListRessources_en

```
/*
test d'utilisation de la classe ListResourceBundle pour
internationaliser une application
13/02/99
*/

import java.util.*;

/**
 * Ressource contenant les traductions anglaises
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 *
 */
public class I18nListRessources_en extends ListResourceBundle {
  public Object[][] getContents() {
    return contents;
  }
```

```
   //tableau des mots clés et des valeurs

   static final Object[][] contents = {
   {"texte_suivant", "Next"},
   {"texte_precedent", "Previous"},
  };
}
```

Exemple (code Java 1.1) : Sources de la classe I18nListRessources_en_US

```
/*
test d'utilisation de la classe ListResourceBundle pour
internationaliser une application
13/02/99
*/
import java.util.*;
/**
* Ressource contenant les traductions américaines
*
* @version 0.10 13 fevrier 1999
* @author Jean Michel DOUDOUX
*
*/

public class I18nListRessources_en_US extends ListResourceBundle {
public Object[][] getContents() {
return contents;
}

//tableau des mots clés et des valeurs

static final Object[][] contents = {
{"texte_suivant", "Next"},
{"texte_precedent", "Previous"},
};
}
```

25.3.5. La création de sa propre classe fille de ResourceBundle

La troisième solution consiste à créer sa propre sous classe de ResourceBundle et à surcharger la méthode handleGetObject().

Exemple (code Java 1.1) :

```
abstract class MesRessources extends ResourceBundle {

   public Object handleGetObject(String cle)  {
      if(cle.equals(" texte_suivant "))
         return " Suivant " ;
      if(cle.equals(" texte_precedent "))
         return "Precedent " ;
         return null ;
   }
}
```

Attention : la classe ResourceBundle contient deux méthodes abstraites : handleGetObjects() et getKeys(). Si l'une des deux n'est pas définie alors il faut définir la sous classe avec le mot clé abstract.

Il faut créer autant de sous classes que de Locale désiré : il suffit simplement d'ajouter dans le nom de la classe le code langue et le code pays avec eventuellement le code variant.

25.3.6. Exemple de classes utilisant une classe fille de ResourceBundle

Exemple (code Java 1.1) : Sources de la classe I18nResource

```
/*
Test d'utilisation d'un sous classement
de la classe ResourceBundle pour
internationaliser une application
13/02/99
*/
import java.util.*;
/**
 * Description de la classe I18nResource
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 */
public class I18nResource {
  /**
   * Constructeur de la classe
   */
  public I18nResource() {

    String texte;
    Locale locale;
    ResourceBundle res;


    System.out.println("Locale par defaut : ");
    res = ResourceBundle.getBundle("I18nResourceBundle");
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);


    System.out.println("\nLocale anglaise : ");
    locale = new Locale("en","");
    res = ResourceBundle.getBundle("I18nResourceBundle", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);

    System.out.println("\nLocale allemande : "+
       "non définie donc utilisation locale par defaut ");
    locale = Locale.GERMAN;
    res = ResourceBundle.getBundle("I18nResourceBundle", locale);
    texte = (String)res.getObject("texte_suivant");
    System.out.println("texte_suivant = "+texte);
    texte = (String)res.getObject("texte_precedent");
    System.out.println("texte_precedent = "+texte);

  }

 /**
   * Pour tester la classe
   *
   * @param      args[]         arguments passes au programme
   */
  public static void main(String[] args) {
    I18nResource i18nResource = new I18nResource();
  }

}
```

Exemple (code Java 1.1) : Sources de la classe I18nResourceBundle

```
/*
Test d'utilisation de la derivation de la classe ResourceBundle pour
internationaliser une application
13/02/99
```

```
*/

import java.util.*;

/**
 * Description de la classe I18nResourceBundle
 * C'est la classe contenant la locale par defaut
 * Contient les traductions de la locale francaise (langue par defaut)
 * Elle herite de ResourceBundle
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 */
public class I18nResourceBundle extends ResourceBundle {
   protected Vector table;

   public I18nResourceBundle() {
      super();
      table = new Vector();
      table.addElement("texte_suivant");
      table.addElement("texte_precedent");
   }

   public Object handleGetObject(String cle)  {
      if(cle.equals(table.elementAt(0))) return "Suivant" ;
      if(cle.equals(table.elementAt(1))) return "Precedent" ;
      return null ;
   }

   public Enumeration getKeys()  {
      return table.elements();
   }

}
```

Exemple (code Java 1.1) : Sources de la classe I18nResourceBundle_en

```
/*
Test d'utilisation de la derivation de la classe ResourceBundle pour
internationaliser une application
13/02/99
*/

import java.util.*;

/**
 * Description de la classe I18nResourceBundle_en
 * Contient les traductions de la locale anglaise
 * Elle herite de la classe contenant la locale par defaut
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 */
public class I18nResourceBundle_en extends I18nResourceBundle {
   public Object handleGetObject(String cle)  {
      if(cle.equals(table.elementAt(0))) return "Next" ;
      if(cle.equals(table.elementAt(1))) return "Previous" ;
      return null ;
   }
}
```

Exemple (code Java 1.1) : Sources de la classe I18nResourceBundle_fr_FR

```
/*
Test d'utilisation de la derivation de la classe ResourceBundle pour
internationaliser une application
13/02/99
*/

import java.util.*;
```

```
/**
 * Description de la classe I18nResourceBundle_fr_FR
 * Contient les traductions de la locale francaise
 * Elle herite de la classe contenant la locale par defaut
 *
 * @version       0.10 13 fevrier 1999
 * @author        Jean Michel DOUDOUX
 */
public class I18nResourceBundle_fr_FR extends I18nResourceBundle {

   /**
    *
    * Retourne toujours null car la locale francaise correspond
    * a la locale par defaut
    *
    */
   public Object handleGetObject(String cle)  {
      return null ;
   }
}
```

26. Les composants java beans

Les java beans sont des composants réutilisables introduits par le JDK 1.1. De nombreuses fonctionnalités de ce JDK lui ont été ajoutées pour développer des caractéristiques de ces composants. Les java beans sont couramment appelés simplement beans.

Les beans sont prévus pour pouvoir inter−agir avec d'autres beans au point de pouvoir développer une application simplement en assemblant des beans avec un outil graphique dédié. Sun fournit gratuitement un tel outil : le B.D.K. (Bean Development Kit).

Ce chapitre contient plusieurs sections :

26.1. Présentations des java beans

Des composants réutilisables sont des objets autonomes qui doivent pouvoir être facilement assemblés entres eux pour créer un programme.

Microsoft propose la technologie ActiveX pour définir des composants mais celle ci est spécifiquement destinée aux plate−formes Windows.

Les Java beans proposés par Sun reposent bien sûre sur java et de fait en possèdent toutes les caractéristiques : indépendance de la plate−forme, taille réduite du composant, ...

La technologie java beans propose de simplifier et faciliter la création et l'utilisation de composants.

Les java beans possèdent plusieurs caractéristiques :

- la persistance : elle permet grâce au mécanisme de sérialisation de sauvegarder l'état d'un bean pour le restaurer ainsi si on assemble plusieurs beans pour former une application, on peut la sauvegarder.
- la communication grâce à des événements qui utilise le modèle des écouteurs introduit par java 1.1
- l'introspection : ce mécanisme permet de découvrir de façon dynamique l'ensemble des éléments qui compose le bean (attributs, méthodes et événements) sans avoir le source.
- la possibilité de paramétrer le composant : les données du paramétrage sont conservées dans des propriétés.

Ainsi, les beans sont des classes java qui doivent respecter un certains nombre de règles :

- ils doivent posséder un constructeur sans paramètre. Celui ci devra initialiser l'état du bean avec des valeurs par

défauts.
- ils peuvent définir des propriétés : celles ci sont identifiées par des méthodes dont le nom et la signature sont normalisés
- ils devraient implémenter l'interface serialisable : ceci est obligatoire pour les beans qui possèdent une partie graphique pour permettre la sauvegarde de leur état
- ils définissent des méthodes utilisables par les composants extérieures : elles doivent être public et prévoir une gestion des accès concurrents
- ils peuvent émettre des événements en gérant une liste d'écouteurs qui s'y abonnent via des méthodes dont les noms sont normalisés

Le type de composants le plus adapté est le composant visuel. D'ailleurs, les composants des classes A.W.T. et Swing pour la création d'interfaces graphiques sont tous des beans. Mais les beans peuvent aussi être des composants non visuels pour prendre en charge les traitements.

26.2. Les propriétés.

Les propriétés contiennent des données qui gèrent l'état du composant : ils peuvent être de type primitif ou être un objet.

Il existe quatres types de propriétés :

- les propriétés simples
- les propriétés indexées (indexed properties)
- les propriétés liées (bound properties)
- les propriétés liées avec contraintes (Constrained properties)

26.2.1. Les propriétés simples

Les propriétés sont des variables d'instance du bean qui possèdent des méthodes particulières pour lire et modifier leur valeur. La normalisation de ces méthodes permet à des outils de déterminer de façon dynamique quelles sont les propriétés du bean. L'accès à ces propriétés doit se faire via ces méthodes. Ainsi la variable qui stocke la valeur de la propriété ne doit pas être déclarée public mais les méthodes d'accès à cette variable doivent bien sûre l'être.

Le nom de la méthode de lecture d'une propriété doit obligatoirement commencer par « get » suivi par le nom de la propriété dont la première lettre doit être une majuscule. Une telle méthode est souvent appelée « getter » ou « accesseur » de la propriété. La valeur retournée par cette méthode doit être du type de la propriété.

Exemple (code Java 1.1) :

```
private int longueur;
public int getLongueur () {
  return longueur;
}
```

Pour les propriétés booléennes, une autre convention peut être utilisée : la méthode peut commencer par «is» au lieu de « get ». Dans ce cas, la valeur de retour est obligatoirement de type boolean.

Le nom de la méthode permettant la modification d'une propriété doit obligatoirement commencer par « set » suivi par le nom de la propriété dont la première lettre doit être une majuscule. Une telle méthode est souvent appelée « setter ». Elle ne retourne aucune valeur et doit avoir en paramètre une variable du type de la propriété qui contiendra sa nouvelle valeur. Elle devra assurer la mise à jour de la valeur de la propriété en effectuant éventuellement des contrôles et/ou des traitements (par exemple le rafraîchissement pour un bean visuel dont la propriété affecte l'affichage).

Exemple (code Java 1.1) :

```
private int longueur ;
public void setLongueur (int longueur) {
  this.longueur = longueur;
```

```
}
```

Une propriété peut n'avoir qu'un getter et pas de setter : dans ce cas, la propriété n'est utilisable qu'en lecture seule.

Le nom de la variable d'instance qui contient la valeur de la propriété n'est pas obligatoirement le même que le nom de la propriété

Il est préférable d'assurer une gestion des accès concurrents dans ces méthodes de lecture et de mise à jour des propriétés par exemple en déclarant ces méthodes synchronized.

Les méthodes du beans peuvent directement manipuler en lecture et écriture la variable d'instance qui stocke la valeur de la propriété, mais il est préférable d'utiliser le getter et le setter.

26.2.2. les propriétés indexées (indexed properties)

Ce sont des propriétés qui possèdent plusieurs valeurs stockées dans un tableau.

Pour ces propriétés, il faut aussi définir des méthodes « get » et « set » dont il convient d'ajouter un paramètre de type int représentant l'index de l'élément du tableau.

Exemple (code Java 1.1) :

```
private float[] notes = new float[5];
public float getNotes (int i ) {
  return notes[i];
}
public void setNotes (int i ; float notes) {
  this.notes[i] = notes;
}
```

Il est aussi possible de définir des méthodes « get » et « set » permettant de mettre à jour tout le tableau.

Exemple (code Java 1.1) :

```
private float[] notes = new float[5] ;
public float[] getNotes () {
  return notes;
}
public void setNotes (float[] notes) {
  this.notes = notes;
}
```

26.2.3. Les propriétés liées (Bound properties)

Il est possible d'informer d'autres composants du changement de la valeur d'une propriété d'un bean. Les java beans peuvent mettre en place un mécanisme qui permet pour une propriété d'enregistrer des composants qui seront informés du changement de la valeur de la propriété.

Ce mécanisme peut être mis en place grâce à un objet de la classe PropertyChangeSupport qui permet de simplifier la gestion de la liste des écouteurs et de les informer des changements de valeur d'une propriété. Cette classe définit les méthodes addPropertyChangeListener() pour enregistrer un composant désirant être informé du changement de la valeur de la propriété et removePropertyChangeListener() pour supprimer un composant de la liste.

La méthode firePropertyChange() permet d'informer tous les composants enregistrés du changement de la valeur de la propriété.

Le plus simple est que le bean hérite de cette classe si possible car les méthodes addPropertyChangeListener() et removePropertyChangeListener() seront directement héritées.

Si ce n'est pas possible, il est obligatoire de définir les méthodes addPropertyChangeListener() et removePropertyChangeListener() dans le bean qui appelleront les méthodes correspondantes de l'objet PropertyChangeSupport.

Exemple (code Java 1.1) :

```
import java.io.Serializable;
import java.beans.*;
public class MonBean03 implements Serializable {
  protected int valeur;

  PropertyChangeSupport changeSupport;

  public MonBean03(){
    valeur = 0;

    changeSupport = new PropertyChangeSupport(this);
  }

  public synchronized void setValeur(int val) {
    int oldValeur = valeur;
    valeur = val;

    changeSupport.firePropertyChange("valeur",oldValeur,valeur);
  }
  public synchronized int getValeur() {
    return valeur;
  }
  public synchronized void addPropertyChangeListener(PropertyChangeListener listener) {
    changeSupport.addPropertyChangeListener(listener);
  }

  public synchronized void removePropertyChangeListener(PropertyChangeListener listener) {
    changeSupport.removePropertyChangeListener(listener);
  }

}
```

Les composants qui désirent être enregistrés doivent obligatoirement implémenter l'interface PropertyChangeListener et définir la méthode propertyChange() déclarée par cette interface.

La méthode propertyChange() reçoit en paramètre un objet de type PropertyChangeEvent qui représente l'événement. Cette méthode de tous les objets enregistrés est appelée. Le paramètre de type PropertyChangeEvent contient plusieurs informations :

- l'objet source : le bean dont la valeur d'une propriété a changé
- le nom de la propriété sous forme de chaîne de caractères
- l'ancienne valeur sous forme d'un objet de type Object
- la nouvelle valeur sous forme d'un objet de type Object

Pour les traitements, il est souvent nécessaire d'utiliser un cast pour transmettre ou utiliser les objets qui représentent l'ancienne et la nouvelle valeur.

Méthode	Rôle
public Object getSource()	retourne l'objet source
public Object getNewValue()	retourne la nouvelle valeur de la propriété
public Object getOldValue()	retourne l'ancienne valeur de la propriété
public String getPropertyName	retourne le nom de la propriété modifiée

Exemple (code Java 1.1) : un programme qui créé le bean et lui associe un écouteur

```
import java.beans.*;
import java.util.*;
```

```
public class TestMonBean03 {
  public static void main(String[] args) {
    new TestMonBean03();
  }

  public TestMonBean03() {
    MonBean03 monBean = new MonBean03();

    monBean.addPropertyChangeListener( new PropertyChangeListener() {
      public void propertyChange(PropertyChangeEvent event) {
        System.out.println("propertyChange : valeur = "+ event.getNewValue());
      }
    } );

    System.out.println("valeur = " + monBean.getValeur());
    monBean.setValeur(10);
    System.out.println("valeur = " + monBean.getValeur());

  }
}
```

Résultat :

```
C:\tutorial\sources exemples>java TestMonBean03
valeur = 0
propertyChange : valeur = 10
valeur = 10
```

Pour supprimer un écouteur de la liste du bean, il suffit d'appeler la méthode removePropertyChangeListener() en lui passant en paramètre un référence sur l'écouteur.

26.2.4. Les propriétés liées avec contraintes (Constrained properties)

Ces propriétés permettent à un ou plusieurs composants de mettre un veto sur la modification de la valeur de la propriété.

Comme pour les propriétés liées, le bean doit gérer un liste de composants « écouteurs » qui souhaitent être informé d'un changement possible de la valeur de la propriété. Si un composant désire s'opposer à ce changement de valeur, il lève une exception pour en informer le bean.

Les écouteurs doivent implémenter l'interface VetoableChangeListener qui définit la méthode vetoableChange().

Avant le changement de la valeur, le bean appelle cette méthode vetoableChange() de tous les écouteurs enregistrés. Elle possède en paramètre un objet de type PropertyChangeEvent qui contient : le bean, le nom de la propriété, l'ancienne valeur et la nouvelle valeur.

Si un écouteur veut s'opposer à la mise à jour de la valeur, il lève une exception de type java.beans.PropertyVetoException. Dans ce cas, le bean ne change pas la valeur de la propriété : ces traitements sont à la charge du programmeur avec notamment la gestion de la capture et du traitement de l'exception dans un bloc try/catch.

La classe VetoableChangeSupport permet de simplifier la gestion de la liste des écouteurs et de les informer du futur changement de valeur d'une propriété. Son utilisation est similaire à celle de la classe PropertyChangeSupport.

Pour ces propriétés, pour que les traitements soient complets il faut implémenter le code pour gérer et traiter les écouteurs qui souhaitent connaître les changements de valeur effectifs de la propriété (voir les propriétés liées).

Exemple (code Java 1.1) :

```
import java.io.Serializable;
import java.beans.*;
public class MonBean04 implements Serializable {
  protected int oldValeur;
  protected int valeur;
```

```
  PropertyChangeSupport changeSupport;
  VetoableChangeSupport vetoableSupport;

  public MonBean04(){
    valeur = 0;
    oldValeur = 0;

    changeSupport = new PropertyChangeSupport(this);
    vetoableSupport = new VetoableChangeSupport(this);
  }


  public synchronized void setValeur(int val) {
    oldValeur = valeur;
    valeur = val;

    try {
      vetoableSupport.fireVetoableChange("valeur",new Integer(oldValeur),
        new Integer(valeur));
    } catch(PropertyVetoException e) {
      System.out.println("MonBean, un veto est emis :  "+e.getMessage());
      valeur = oldValeur;
    }
    if ( valeur != oldValeur ) {
      changeSupport.firePropertyChange("valeur",oldValeur,valeur);
    }

  }
  public synchronized int getValeur() {
    return valeur;
  }
  public synchronized void addPropertyChangeListener(PropertyChangeListener listener) {
    changeSupport.addPropertyChangeListener(listener);
  }

  public synchronized void removePropertyChangeListener(PropertyChangeListener listener) {
    changeSupport.removePropertyChangeListener(listener);
  }
  public synchronized void addVetoableChangeListener(VetoableChangeListener listener) {
    vetoableSupport.addVetoableChangeListener(listener);
  }
  public synchronized void removeVetoableChangeListener(VetoableChangeListener listener) {
    vetoableSupport.removeVetoableChangeListener(listener);
  }

}
```

Exemple (code Java 1.1) : un programme qui teste le bean. Il émet un veto si la nouvelle valeur de la propriété est supérieure à 100.

```
import java.beans.*;
import java.util.*;
public class TestMonBean04 {
  public static void main(String[] args) {
    new TestMonBean04();
  }

  public TestMonBean04() {
    MonBean04 monBean = new MonBean04();

    monBean.addPropertyChangeListener( new PropertyChangeListener() {
      public void propertyChange(PropertyChangeEvent event) {
        System.out.println("propertyChange : valeur = "+ event.getNewValue());
      }
    } );

    monBean.addVetoableChangeListener( new VetoableChangeListener() {

      public void vetoableChange(PropertyChangeEvent event) throws PropertyVetoException {
        System.out.println("vetoableChange : valeur = " + event.getNewValue());
        if( ((Integer)event.getNewValue()).intValue() > 100 )
          throw new PropertyVetoException("valeur superieur a 100",event);
      }
```

```
    } );

    System.out.println("valeur = " + monBean.getValeur());
    monBean.setValeur(10);
    System.out.println("valeur = " + monBean.getValeur());
    monBean.setValeur(200);
    System.out.println("valeur = " + monBean.getValeur());

  }
}
```

Résultat :

```
C:\tutorial\sources exemples>java TestMonBean04
valeur = 0
vetoableChange : valeur = 10
propertyChange : valeur = 10
valeur = 10
vetoableChange : valeur = 200
vetoableChange : valeur = 10
MonBean, un veto est emis :  valeur superieur a 100
valeur = 10
```

26.3. Les méthodes

Toutes les méthodes publiques sont visibles de l'extérieur et peuvent donc être appelées.

26.4. Les événements

Les beans utilisent les événements définis dans le modèle par délégation introduit par le J.D.K. 1.1 pour dialoguer. Par respect de ce modèle, le bean est la source et les autres composants qui souhaitent être informés sont nommés « Listeners » ou « écouteurs » et doivent s'enregistrer auprès du bean qui maintient la liste des composants enregistrés.

Il est nécessaire de définir les méthodes qui vont permettre de gérer la liste des écouteurs désirant recevoir l'événement. Il faut définir deux méthodes :

- public void addXXXListener(XXXListener li) pour enregistrer l'écouteur li
- public void removeXXXListener (XXXListener li) pour enlever l'écouteur li de la liste

L'objet de type XXXListener doit obligatoirement implémenter l'interface java.util.EventListener et son nom doit terminer par « Listener ».

Les événements peuvent être mono ou multi écouteurs.

Pour les évenements mono écouteurs, la méthode addXXXListener() doit indiquer dans sa signature qu'elle est succeptible de lever l'exception java.util.TooManyListenersException si un écouteur tente de s'enregistrer et qu'il y en a déjà un présent.

26.5. L'introspection

L'introspection est un mécanisme qui permet de déterminer de façon dynamique les caractéristiques d'une classe et donc d'un bean. Les caractéristiques les plus importantes sont les propriétés, les méthodes et les événements. Le principe de l'introspection permet à Sun d'éviter de rajouter des éléments au langage pour définir ces caractéristiques.

L'API JavaBean définit la classe java.beans.Introspector qui facilite et standardise la recherche des propriétés, méthodes et événements du bean. Cette classe possède des méthodes pour analyser le bean et retourner un objet de type BeanInfo contenant les informations trouvées.

La classe Introspector utilise deux techniques pour retrouver ces informations :

1. un objet de type BeanInfo, si il y en a un défini par les développeurs du bean
2. les mécanismes fournis par l'API reflexion pour extraire les entités qui respectent leur modèle (design pattern) respectif.

Il est donc possible de définir un objet BeanInfo qui sera directement utilisé par la classe Introspector. Cette définition est utile si le bean ne respecte pas certains modèles (designs patterns) ou si certaines entités héritées ne doivent pas être utilisables. Dans ce cas, le nom de cette classe doit obligatoirement respecter le modèle XXXBeanInfo ou XXX est le nom du bean correspondant. La classe Introspector recherche une classe respectant ce modèle.

Si une classe BeanInfo pour un bean est définie, une classe qui hérite du bean n'est pas obligée de définir un classe BeanInfo. Dans ce cas, la classe Introspector utilise les informations du BeanInfo de la classe mère et ajoute les informations retournées par l'API reflection sur le bean.

Sans classe BeanInfo associée au bean, les méthodes de la classe Introspector utilisent les techniques d'introspection pour analyser le bean.

26.5.1. Les modèles (designs patterns)

Si la classe Introspector utilise l'API reflection pour déterminer les informations sur le bean et utilise en même temps un ensembles de modèles sur chacunes des entités propriétés, méthodes et événements.

Pour déterminer les propriétés, la classe Introspector recherche les méthodes getXxx, setXxx et isXxx ou Xxx représente le nom de la propriété dont la première lettre est en majuscule. La première lettre du nom de la propriété est remise en minuscule sauf si les deux premières lettres de la propriété sont en majuscules.

Pour déterminer les méthodes, la classe Introspector rechercher toutes les méthodes publiques.

Pour déterminer les événements, la classe Introspector recherche les méthodes addXxxListener() et removeXxxListener(). Si les deux sont présentes, elle en déduit que l'événement xxx est défini dans le bean. Comme pour les propriétés, la première lettre du nom de l'événement est mis en minuscule.

26.5.2. La classe BeanInfo

La classe BeanInfo contient des informations sur un bean et possède plusieurs méthodes pour les obtenir.

La méthode getBeanInfo() prend en paramètre un objet de type Class qui représente la classe du bean et elle renvoie des informations sur la classe et toutes ses classes mères.

Une version surchargée de la méthode accepte deux objets de type Class : le premier représente le bean et le deuxième représente une classe appartenant à la hiérarchie du bean. Dans ce cas, la recherche d'informations d'arrêtera juste avant d'arriver à la classe précisée en deuxième argument.

Exemple : obtenir des informations sur le bean uniquement (sans informations sur ces super classes)

Exemple (code Java 1.1) :

```
Class monBeanClasse = Class.forName("monBean");
BeanInfo bi = Introspector.getBeanInfo(monBeanClasse, monBeanClasse.getSuperclass());
```

La méthode getBeanDescriptor() permet d'obtenir des informations générales sur le bean en renvoyant un objet de type BeanDescriptor()

La méthode getPropertyDescriptors() permet d'obtenir un tableau d'objets de type PropertyDescriptor qui contient les caractéristiques d'une propriété. Plusieurs méthodes permettent d'obtenir ces informations.

Exemple (code Java 1.1) :

```
    PropertyDescriptor[] propertyDescriptor = bi.getPropertyDescriptors();
    for (int i=0; i<propertyDescriptor.length; i++) {
      System.out.println(" Nom propriete    : " +
                         propertyDescriptor[i].getName());
      System.out.println(" Type propriete   : "
                + propertyDescriptor[i].getPropertyType());
      System.out.println(" Getter propriete : "
                  + propertyDescriptor[i].getReadMethod());
      System.out.println(" Setter propriete : "
                 + propertyDescriptor[i].getWriteMethod());
    }
```

La méthode getMethodDescriptors() permet d'obtenir un tableau d'objets de type MethodDescriptor qui contient les caractéristiques d'une méthode. Plusieurs méthodes permettent d'obtenir ces informations.

Exemple (code Java 1.1) :

```
   MethodDescriptor[] methodDescriptor;
    unMethodDescriptor = bi.getMethodDescriptors();
    for (int i=0; i < unMethodDescriptor.length; i++) {
      System.out.println(" Methode : "+unMethodDescriptor[i].getName());
    }
```

La méthode getEventSetDescriptors() permet d'obtenir un tableau d'objets de type EventSetDescriptor qui contient les caractéristiques d'un événement. Plusieurs méthodes permettent d'obtenir ces informations.

Exemple (code Java 1.1) :

```
    EventSetDescriptor[] unEventSetDescriptor = bi.getEventSetDescriptors();
    for (int i = 0; i < unEventSetDescriptor.length; i++) {
      System.out.println(" Nom evt             : "
                        + unEventSetDescriptor[i].getName());
      System.out.println(" Methode add evt     : " +
             unEventSetDescriptor[i].getAddListenerMethod());
      System.out.println(" Methode remove evt  : " +
          unEventSetDescriptor[i].getRemoveListenerMethod());
      unMethodDescriptor = unEventSetDescriptor[i].getListenerMethodDescriptors();
      for (int j = 0; j < unMethodDescriptor.length; j++) {
          System.out.println(" Event Type: " + unMethodDescriptor[j].getName());
      }
    }
```

Exemple complet (code Java 1.1) :

```
import java.io.*;
import java.beans.*;
import java.lang.reflect.*;
public class BeanIntrospection {
  static String nomBean;
  public static void main(String args[]) throws Exception {
    nomBean = args[0];
    new BeanIntrospection();
  }

  public BeanIntrospection() throws Exception {
    Class monBeanClasse = Class.forName(nomBean);
    MethodDescriptor[] unMethodDescriptor;

    BeanInfo bi = Introspector.getBeanInfo(monBeanClasse, monBeanClasse.getSuperclass());
    BeanDescriptor unBeanDescriptor = bi.getBeanDescriptor();
    System.out.println("Nom du bean    : " + unBeanDescriptor.getName());
    System.out.println("Classe du bean : " + unBeanDescriptor.getBeanClass());
    System.out.println("");

    PropertyDescriptor[] propertyDescriptor = bi.getPropertyDescriptors();
    for (int i=0; i<propertyDescriptor.length; i++) {
```

```
      System.out.println(" Nom propriete    : " +
                         propertyDescriptor[i].getName());
      System.out.println(" Type propriete   : "
                + propertyDescriptor[i].getPropertyType());
      System.out.println(" Getter propriete : "
                 + propertyDescriptor[i].getReadMethod());
      System.out.println(" Setter propriete : "
                 + propertyDescriptor[i].getWriteMethod());
    }
    System.out.println("");
    unMethodDescriptor = bi.getMethodDescriptors();
    for (int i=0; i < unMethodDescriptor.length; i++) {
      System.out.println(" Methode : "+unMethodDescriptor[i].getName());
    }
    System.out.println("");
    EventSetDescriptor[] unEventSetDescriptor = bi.getEventSetDescriptors();
    for (int i = 0; i < unEventSetDescriptor.length; i++) {
      System.out.println(" Nom evt             : "
                       + unEventSetDescriptor[i].getName());
      System.out.println(" Methode add evt     : " +
             unEventSetDescriptor[i].getAddListenerMethod());
      System.out.println(" Methode remove evt  : " +
          unEventSetDescriptor[i].getRemoveListenerMethod());
      unMethodDescriptor = unEventSetDescriptor[i].getListenerMethodDescriptors();
      for (int j = 0; j < unMethodDescriptor.length; j++) {
          System.out.println(" Event Type: " + unMethodDescriptor[j].getName());
      }
    }
    System.out.println("");
  }
}
```

26.6. Paramétrage du bean (Customization)

Il est possible de développer un éditeur de propriétés spécifique pour permettre de personnaliser la modification des paramètres du bean.

La suite de cette section sera développée dans une version future de ce document

26.7. La persistance

Les propriétés du bean doivent pouvoir être sauvés pour être restitués ultérieurement. Le mécanisme utilisé est la sérialisation. Pour permettre d'utiliser ce mécanisme, le bean doit implémenter l'interface Serializable

26.8. La diffusion sous forme de jar

Pour diffuser un bean sous forme de jar, il faut définir un fichier manifest.

Ce fichier doit obligatoirement contenir un attribut Name: qui contient le nom complet de la classe (incluant le package) et un attribut Java−Bean: valorisé à True.

Exemple de fichier manifest pour un bean :

```
Name: MonBean.class
 Java-Bean: True
```

La suite de cette section sera développée dans une version future de ce document

26.9. Le B.D.K.

L'outil principal du B.D.K. est la BeanBox. Cet outil écrit en java permet d'assembler des beans.

Pour utiliser un bean dans la BeanBox, il faut qu'il soit utilisé sous forme d'archive jar.

La suite de cette section sera développée dans une version future de ce document

27. Logging

Le logging est important dans toutes les applications pour permettre de faciliter le débogage lors du développement et de conserver une trace de son exécution lors de l'exploitation en production.

Une API très répandue est celle développée par le projet open source log4j du groupe Jakarta.

Concient de l'importance du logging, Sun a développé et intégré au JDK 1.4 une API dédiée. Cette API est plus simple à utiliser et elle offre moins de fonctionnalités que log4j mais elle a l'avantage d'être fournie directement avec le JDK.

Face au dilemme du choix de l'utilisation de ces deux API, le groupe Jakarta a développé qui permet d'utiliser indifférement l'une ou l'autre de ces deux API.

Ce chapitre contient plusieurs sections :

27.1. Log4j

Log4j est un projet open source distribué sous la licence Apache Software. Cette API permet aux développeurs d'utiliser et de paramétrer les traitements de logs. Il est possible de fournir les paramètres de l'outil dans un fichier de configuration. Log4j est compatible avec le JDK 1.1. et supérieur.

L'avantage de configurer les traitements de log4j dans un fichier externe est de pouvoir modifier la configuration des traitements de logging sans avoir à modifier le code source. Log4j gère plusieurs niveaux de gravités et les messages peuvent être envoyés dans plusieurs flux : un fichier sur disque, le journal des evénements de Windows, une connexion TCP/IP, une base de données, un canal JMS, ...

La dernière version de log4j est téléchargeable à l'url http://jakarta.apache.org/log4j/ : elle inclus les binaires (les fichiers .class), les sources complètes et la documentation.

Log4j utilise trois composants principaux :

- Catégories : ils permettent de gérer les logs
- Appenders : ils représentent les flux qui vont recevoir les messages de log
- Layouts : ils permettent de formatter le contenu des messages du fichier de log

Log4j est thread−safe.

27.1.1. Les catégories

Les catégories déterminent si un message doit être envoyé dans le ou les logs. Elles sont représentées par la classe org.apache.log4j.Category

Chaque catégorie possède un nom qui est sensible à la casse. Pour créer une catégorie, il suffit d'instancier un objet Category. Pour réaliser cette instanciation, la classe Category fournie une méthode statique getInstance() qui attend en paramètre le nom de la catégorie. Si une catégorie existe déjà avec le nom fourni, alors la méthode getInstance() renvoie une instance sur cette catégorie.

Il est pratique de fournir le nom complet de la classe comme nom de la catégorie dans laquelle elle est instanciée mais ce n'est pas obligation. Il est ainsi possible de créer une hiérarchie spécifique différente de celle de l'application, par exemple basée sur des aspects fonctionnels. L'inconvénient d'associer le nom de la classe au nom de la catégorie est qu'il faut instancier un objet Category dans chaque classe. Le plus pratique est de déclarer cet objet static.

Exemple :

```
public class Classe1 {
  static Category category = Category.getInstance(Classe1.class.getName());
  ...
}
```

La méthode log(Priority, Object) permet de demander l'envoie dans le log du message avec la priorité fournie.

Il existe en plus une méthode qui permet de demander l'envoi d'un message dans le fichier de log pour chaque priorité : debug(Object), info(Object), warn(Object), error(Object), fatal(Object).

La demande est traitée en fonction de la hiérarchie de la catégorie et de la priorité du message.

Pour éviter d'éventuels traitements pour créer le message, il est possible d'utiliser la méthode isEnabledFor(Priority) pour savoir si la catégorie prend en compte la priorité ou non.

Exemple :

```
import org.apache.log4j.*;

public class TestIsEnabledFor {

  static Category cat1 = Category.getInstance(TestIsEnabledFor.class.getName());

  public static void main(String[] args) {
    int i=1;
    int[] occurence={10,20,30};

    BasicConfigurator.configure();

    cat1.setPriority(Priority.WARN) ;
    cat1.warn("message de test");

    if(cat1.isEnabledFor(Priority.INFO)) {
      System.out.println("traitement du message de priorité INFO");
      cat1.info("La valeur de l'occurence "+i+" = " + String.valueOf(occurence[i]));
    }
    if(cat1.isEnabledFor(Priority.WARN)) {
      System.out.println("traitement du message de priorité WARN");
      cat1.warn("La valeur de l'occurence "+i+" = " + String.valueOf(occurence[i]));
    }

  }
}
```

Résultat :

```
0 [main] WARN TestIsEnabledFor  − message de test
traitement du message de priorit_ WARN
50 [main] WARN TestIsEnabledFor  − La valeur de l'occurence 1 = 20
```

27.1.1.1. La hiérarchie dans les catégories

Le nom de la catégorie permet de la placer dans une hierarchie dont la racine est une catégorie spéciale nommée root qui est créée par défaut sans nom.

La classe Category possède une classe statique getRoot() pour obtenir la catégorie racine.

La hiérarchie des noms est établie grace à la notation par point comme pour les packages.

Exemple : soit trois catégories
categorie1 nommée "org"
categorie2 nommée "org.moi"
categorie3 nommée "org.moi.projet"

Categorie3 est fille de categorie2, elle même fille de categorie1.

Cette relation hiérarchique est importante car la configuration établie pour une catégorie est automatiquement propagée par défaut aux catégories enfants.

L'ordre de la création des catégories de la hiérarchie ne doit pas obligatoirement respecter l'ordre de la hiérarchie. Celle ci est constituée au fur et à mesure de la création des catégories.

27.1.1.2. Les priorités

Log4j gére des priorités pour permettre à la catégorie de déterminer si le message sera envoyé dans le fichier de log. Il existe cinq priorités qui possèdent un ordre hiérarchique croissant :

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

La classe org.apache.log4j.Priority encapsule ces priorités.

Chaque catégorie est associée à une priorité qui peut être changée dynamiquement. La catégorie détermine si un message doit être envoyé dans le fichier de log en comparant sa priorité avec la priorité du message. Si celle est supérieure ou égale à la priorité de la catégorie, alors le message est envoyé dans le fichier log.

La méthode setPropriety() de la classe Category permet de préciser la priorité de la catégorie.

Si aucune priorité n'est donnée à une catégorie, elle "hérite" de la priorité de la première catégorie en remontant dans la hiérarchie dont la priorité est renseignée.

Exemple : soit trois catégories
root associée à la priorité INFO
categorie1 nommée "org" sans priorité particulière
categorie2 nommée "org.moi" associée à la priorité ERROR
categorie3 nommée "org.moi.projet" sans priorité particulière

Une demande avec la priorité DEBUG sur categorie2 n'est pas traitée car la priorité INFO héritée est supérieure à DEBUG.
Une demande avec la priorité WARN sur categorie2 est traitée car la priorité INFO héritée est inférieure à WARN .
Une demande avec la priorité DEBUG sur categorie3 n'est pas traitée car la priorité ERROR héritée est supérieure à DEBUG.
Une demande avec la priorité FATAL sur categorie3 est traitée car la priorité ERROR héritée est inférieure à FATAL.

En fait dans l'exemple, aucune demande avec la priorité DEBUG ne sera traitée.

Au niveau applicatif, il est possible d'interdire le traitement d'une priorité et de celle inférieure en utilisant le code suivant : Category.getDefaultHierarchy().disable(). Il faut fournir la priorité à la méthode disable().

Il est possible d'annuler ce traitement dynamiquement en positionnement la propriété système : log4j.disableOverride.

27.1.2. Les Appenders

L'interface org.apache.log4j.Appender désigne un flux qui représente le fichier de log et se charge de l'envoie de message formatté ce flux. Le formattage proprement dit est réalisé par un objet de type Layout. Ce layout peut être fourni dans le constructeur adapté ou par la méthode setLayout().

Une catégorie peut posséder plusieurs appenders. Si la catégorie décide de traiter la demande de message, le message est envoyés à chacun des appenders. Pour ajouter un appender à un catégorie, il suffit d'utiliser la méthode addAppender() qui attend en paramètre un objet de type Appender.

L'interface Appender est directement implémentée par la classe abstraite AppenderSkeleton. Cette classe est la classe mère de toutes les classes fournies avec log4j pour représenter un type de log :

- AsyncAppender
- JMSAppender
- NTEventLogAppender : log envoyée dans le fichier de log des évenements sur un système Windows NT
- NullAppender
- SMTPAppender
- SocketAppender : log envoyées dans une socket
- SyslogAppender : log envoyée dans le demon syslog d'un système Unix
- WriterAppender : cette classe possède deux classes filles : ConsoleAppender et FileAppender. La classe FileAppender possède deux classes filles : DailyRollingAppender, RollingFileAppender

Pour créer un appender, il suffit d'instancier un objet d'une de ces classes.

Tout comme les priorités, les appenders d'une catégorie mère sont héritées par les catégories filles. Pour éviter cette héritage par défaut, il faut mettre le champ additivity à false en utilisant la méthode setAdditivity() de la classe Category.

27.1.3. Les layouts

Ces composants représentés par la classe org.apache.log4j.Layout permettent de définir le format du fichier de log. Un layout est associé à un appender lors de son instanciation.

Il existe plusieurs layouts définis par log4j :

- DateLayout
- HTMLLayout
- PatternLayout
- SimpleLayout
- XMLLayout

Le PatternLayout permet de préciser le format du fichier de log grâce à des motifs qui sont dynamiquement remplacés par leur valeur à l'exécution. Les motifs commencent par un caractère % suivi d'une lettre :

Motif	Role
%c	le nom de la catégorie qui a émis le message
%C	le nom de la classe qui a émis le message
%d	le timestamp de l'émission du message

%m	le message
%n	un retour chariot
%p	la priorité du message
%r	le nombre de milliseconde écoulé entre le lancement de l'application et l'émission du message
%t	le nom du thread
%x	NDC du thread
%%	le caractère %

Il est possible de préciser le formattage de chaque motif grâce à un alignement et/ou une troncature.Dans le tableau ci dessous, la caractère # représente une des lettres du tableau ci dessus, n représente un nombre de caractères.

Motif	Role
%#	aucun formattage (par défaut)
%n#	alignement à droite, des blancs sont ajoutés si la taille du motif est inférieure à n caractères
%−n#	alignement à gauche, des blanc sont ajoutés si la taille du motif est inférieure à n caractères
%.n	tronque le motif si il est supérieur à n caractères
%−n.n#	alignement à gauche, taille du motif obligatoirement de n caractères (troncature ou complément avec des blancs)

27.1.4. Le fichier de configuration

Le fichier de configuration de permet basiquement de définir le niveau de gravité minimum à traiter, les flux de sorties (Appender) et le format des messages (Layout).

Ce fichier peut utiliser deux formats :

- un fichier .properties utilisant des paires clé/valeur
- un fichier XML

27.1.5. La configuration

Pour faciliter la configuration de log4j, l'API fournie plusieurs classes qui implémentent l'interface Configurator. La classe BasicConfgurator est le classe mère des classes PropertyConfigurator (pour la configuration via un fichier de propriétés) et DOMConfigurator (pour la configuration via un fichier XML).

La classe BasicConfigurator permet de configurer la catégorie root avec des valeurs par défaut. L'appel à la méthode configure() ajoute à la catégorie root la priorité DEBUG et un ConsoleAppender vers la sortie standard (System.out) associé à un PatternLayout (TTCC_CONVERSION_PATTERN qui est une constante défnie dans la classe PatternLayout).

Exemple :

```
import org.apache.log4j.*;

public class TestBasicConfigurator {
  static Category cat = Category.getInstance(TestBasicConfigurator.class.getName()) ;

  public static void main(String[] args) {
    BasicConfigurator.configure();
```

```
    cat.info("Mon message");
  }
}
```

Résultat :

```
0 [main] INFO TestBasicConfigurator  − Mon message
```

La classe PropertyConfigurator permet de configurer log4j à partir d'un fichier de propriétés ce qui évite la recompilation de classes pour modifier la configuration. La méthode configure() qui attend un nom de fichier permet de charger la configuration.

Exemple :

```
import org.apache.log4j.*;

public class TestLogging6 {
   static Category cat = Category.getInstance(TestLogging6.class.getName());

   public static void main(String[] args) {
      PropertyConfigurator.configure("logging6.properties");
      cat.info("Mon message");
   }
}
```

Exemple : le fichier loggin6.properties de configuration de log4j

```
# Affecte a la catégorie root la priorité DEBUG et un appender nommé CONSOLE_APP
log4j.rootCategory=DEBUG, CONSOLE_APP
# le appender CONSOL_APP est associé à la console
log4j.appender.CONSOLE_APP=org.apache.log4j.ConsoleAppender
# CONSOLE_APP utilise un PatternLayout qui affiche : le nom du thread, la priorité,
# le nom de la catégorie et le message
log4j.appender.CONSOLE_APP.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE_APP.layout.ConversionPattern= [%t] %p %c − %m%n
```

Résultat :

```
C:\>java TestLogging6
[main] INFO TestLogging6 − Mon message
```

La classe DOMConfigurator permet de configurer log4j à partir d'un fichier XML ce qui évite aussi la recompilation de classes pour modifier la configuration. L méthode configure() qui attend un nom de fichier permet de charger la configuration. Cette méthode nécessite un parser XML de type DOM compatible avec l'API JAXP.

Exemple :

```
import org.apache.log4j.*;
import org.apache.log4j.xml.*;

public class TestLogging7 {
   static Category cat = Category.getInstance(TestLogging7.class.getName());

   public static void main(String[] args) {
      try {
         DOMConfigurator.configure("logging7.xml");
      } catch (Exception e) {
         e.printStackTrace();
      }
      cat.info("Mon message");
   }
}
```

Exemple : le fichier loggin7.xml de configuration de log4j

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="CONSOLE_APP" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="[%t] %p %c - %m%n"/>
    </layout>
  </appender>
  <root>
    <priority value ="DEBUG" />
    <appender-ref ref="CONSOLE_APP" />
  </root>
</log4j:configuration>
```

Résultat :

```
C:\j2sdk1.4.0-rc\bin>java TestLogging7
[main] INFO TestLogging7 - Mon message
```

27.2. L'API logging

L'usage de fonctionnalités de logging dans les applications est tellement répandu que SUN a décidé de développer sa propre API et de l'intégrer au JDK à partir de la version 1.4.

Cette API a été proposée à la communauté sous la Java Specification Request numéro 047 (JSR-047).

Le but est de proposer un système qui puisse être exploité facilement par toutes les applications.

L'API repose sur cinq classes principales et une interface:

- Logger : cette classe permet d'envoyer des messages sans le système de log
- LogRecord : cette classe encapsule le message
- Handler : cette classe représente la destination qui va recevoir les messages
- Formatter : cette classe permet de formatter le message avant son envoi vers la destination
- Filter : cette interface doit être implémentée par les classes dont le but est de déterminer si le message doit être envoyé vers une destination
- Level : cette représente le niveau de gravité du message

Un logger possède un ou plusieurs Handlers qui sont des entités qui vont recevoir les messages. Chaque handler peut avoir un filtre associé en plus du filtre associé au Logger.

Chaque message possède un niveau de sévérité représenté par la classe Level.

27.2.1. La classe LogManager

Cette classe est un singleton qui propose la méthode getLogManager() pour obtenir l'unique référence sur un objet de ce type.

Cette objet permet :

- de maintenir une liste de Logger désigné par un nom unique.
- de maintenir une liste de Handlers globaux
- de modifier le niveau de sévérité pris en compte pour un ou plusieurs Logger dont le début du nom est précisé

Pour réaliser ces actions, la classe LogManager possède plusieurs méthodes dont les principales sont :

Méthode	Rôle
void addGlobalHandler(Handler)	Ajoute un handler à la liste des handler globaux

Logger getLogger(String)	Permet d'ajouter un nouveau Logger dont le nom fourni en paramère lui sera attribué si il n'existe pas encore sinon la référence sur le Logger qui possède déjà ce nom est renvoyé.
void removeAllGlobalHandlers()	Supprime tous les Handler de la listede handler globaux
void removeGlobalHandler(Handler)	Supprime le Handler de la liste de Handler globaux
void setLevel(String, Level)	Permet de préciser le niveau de tout les Logger dont le nom commence par la chaîne fournie en paramètre
LogManager getLogManager()	Renvoie l'instance unique de cette classe

Par défaut, la liste des Loggers contient toujours un Logger nommé global qui peut être facilement utilisé.

27.2.2. La classe Logger

La classe Logger est la classe qui se charge d'envoyer les messages dans la log. Un Logger est identifié par un nom qui est habituellement le nom qualifié de la classe dans laquelle le Logger est utilisé. Ce nom permet de gérer une hiérarchie de Logger. Cette gestion est assurée par le LogManager. Cette hiérarchie permet d'appliquer des modifications sur un Logger ainsi qu'à toute sa "descendance".

Il est aussi de possible de créer des Logger anonymes.

La méthode getLogger() de la classe LogManager permet d'instancier un nouvel objet Logger si aucun Logger possédant le nom passé en paramètre a déjà été défini sinon il renvoie l'instance existante.

La classe Logger se charge d'envoyer les messages aux Handlers enregistrés sous la forme d'un objet de type LogRecord. Par défaut, ces Handlers sont ceux enregistrés dans le LogManager. L'envoi des messages est conditionné par la comparaison du niveau de sévérité de message avec celui associé au Logger.

La classe Logger possède de nombreuses méthodes pour générer des messages : plusieurs méthodes sont définies pour chaque niveau de sécurité. Plutôt que d'utiliser la méthode log() en précisant le niveau de sévérité, il est possible d'utiliser la méthode correspondante au niveau de sécurité.

Ces méthodes sont surchargées pour accepter plusieurs paramètres :

- le nom de la classe, le nom de la méthode et le message : les deux premières données sont très utiles pour faciliter le debuggage
- le message: dans ce cas, le framework tente de déterminer dynamiquement le nom de la classe et de la méthode

27.2.3. La classe Level

Chaque message est associé à un niveau de sévérité représenté par un objet de type Level. Cette classe définit 7 niveaux de sévérité :

- Level.SEVERE
- Level.WARNING
- Level.INFO
- Level.CONFIG
- Level.FINE
- Level.FINER
- Level.FINEST

27.2.4. La classe LogRecord

27.2.5. La classe Handler

Le framework propose plusieurs classes filles qui représentent différents moyens pour émettre les messages :

- StreamHandler : envoie des messages dans un flux de sortie
- ConsoleHandler : envoie des messages sur la sortie standard d'erreur
- FileHandler : envoie des messages sur un fichier
- SocketHandler : envoie des messages dans une socket réseau
- MemoryHandler : envoie des messages dans un tampon en mémoire

27.2.6. La classe Filter

27.2.7. La classe Formatter

Le framework propose deux implémentations :

- SimpleFormatter : pour formatter l'enregistrement sous forme de chaîne de caractères
- XMLFormatter : pour formatter l'enregistrement au format XML

XMLFormatter utilise un DTD particulière. Le tag racine est <log>. Chaque enregistrement est inclus dans un tag <record>.

27.2.8. Le fichier de configuration

Un fichier particulier au format Properties permet de préciser des paramètres de configuration pour le système de log tel que le niveau de sévérité géré par un Logger particulier et sa descendance, les paramètres de configuration des Handlers ...

Il est possible de préciser le niveau de sévérité pris en compte par tous les Logger :

.level = niveau

Il est possible de définir les handlers par défaut :

handlers = java.util.logging.FileHandler

Pour préciser d'autre handler, il faut les séparer par des virgules.

Pour préciser le niveau de sévérité d'un Handler, il suffit de le lui préciser :

java.util.logging.FileHandler.level = niveau

Un fichier par défaut est défini avec les autres fichiers de configuration dans le répertoire lib du JRE. Ce fichier ce nomme logging.properties.

Il est possible de préciser un fichier particulier précisant son nom dans la propriété système java.util.logging.config.file

exemple : java −Djava.util.logging.config.file=monLogging.properties

27.2.9. Exemples d'utilisation

Cette section sera développée dans une version future de ce document

27.3. Jakarta Commons Logging (JCL)

Le projet Jakarta Commons propose un sous projet nommé Logging qui encapsule l'usage de plusieurs systèmes de logging et facilite ainsi leur utilisation dans les applications. Ce n'est pas un autre système de log mais il propose un niveau d'abstraction qui permet sans changer le code d'utiliser indifféremment n'importe lequel des systemes de logging supportés. Son utilisation est d'autant plus pratique qu'il existe plusieurs système de log dont aucun des plus répandus, Log4j et l'API logging du JDK 1.4, ne sont dominants.

Le grand intérêt de cette bibliothèque est donc de rendre l'utilisation d'un systeme de log dans le code indépendant de l'implémentation de ce système. JCL encapsule l'utilisation de Log4j, l'API logging du JDK 1.4 et LogKit.

De nombreux projets du groupe Jakarta utilise cette bibliothèque tel que Tomcat ou Struts. La version de JCL utilisée dans cette section est le 1.0.3

Le package, contenu dans le fichier commons-logging-1.0.3.zip peut être téléchargé sur le site http://jakarta.apache.org/commons/logging.html. Il doit ensuite être dézippé dans un répertoire du système d'exploitation.

Pour utiliser la bibliothèque, il faut ajouter le fichier dans le classpath.

L'inconvénient d'utiliser cette bibliothèque est qu'elle n'utilise que le dénominateur commun des système de log qu'elle supporte : ainsi certaines caractéristiques d'un système de log particulier ne pourront être utilisées via cette API .

La bibliotèque propose une fabrique qui renvoie, en fonction du paramètre précisé, un objet qui implémente l'interface Log. La méthode statique getLog() de la classe LogFactory permet d'obtenir cet objet : elle attend en paramètre soit un nom sous la forme d'une chaîne de caractères soit un objet de type Class dont le nom sera utilisé. Si un objet de type log possédant ce nom existe déjà alors c'est cette instance qui est renvoyée par la méthode sinon c'est une nouvelle instance qui est retournée. Ce nom représente la catégorie pour le système log utilisé, si celui ci supporte une telle fonctionnalité.

Par défaut, la méthode getLog() utilise les règles suivantes pour déterminer le système de log à utiliser :

- Si la bibliothèque Log4j est incluse dans le classpath de la JVM alors celle ci sera utilisée par défaut par la biliothèque Commons Logging.
- Si le JDK 1.4 est utilisé et que Log4j n'est pas trouvé alors le système utilisé par défaut est celui fourni en standard avec le JDK (java.util.logging)
- Si aucun de ces systèmes de log n'est trouvé, alors JCL utilise un système de log basic fourni dans la bibliothèque : SimpleLog. La configuration de ce système ce fait dans un fichier nommé simplelog.properties

Il est possible de forcer l'usage d'un système de log particulier en précisant la propriété org.apache.commons.logging.Log à la machine virtuelle.

Pour complétement désactiver le système de log, il suffit de fournir la valeur org.apache.commons.logging.impl.NoOpLog pour la propriété org.apache.commons.logging.Log à la JVM. Attention dans ce cas, plus aucune log ne sera émise.

Il existe plusieurs niveaux de gravité que la bibliothèque tentera de faire correspondre au mieux avec le système de log utilisé.

27.4. D'autres API de logging

Il existe d'autres API de logging dont voici une liste non exhaustive :

Produit	URL
Lumberjack	http://javalogging.sourceforge.net/
Javalog	http://sourceforge.net/projects/javalog/
Jlogger de Javelin Software	http://www.javelinsoft.com/jlogger/

28. La sécurité

La suite de ce chapitre sera développée dans une version future de ce document

Depuis sa conception, la sécurité dans le langage Java a toujours été une grande préoccupation pour Sun.

Avec Java, la sécurité revêt de nombreux aspects :

- les spécifications du langage dispose de fonctionnalité pour renforcer la sécurité du code
- la plate−forme définit un modèle pour gérer les droits d'une application
- l'API JCE permet d'utiliser des technologies de cryptographie
- l'API JSSE permet d'utiliser le réseau au travers des protocoles sécurisés SSL ou TLS
- l'API JAAS propose un service pour gérer l'authentification et les autorisations d'un utilisateur

Ces deux premiers aspects ont été intégrés à java dès sa première version.

Ce chapitre contient plusieurs sections :

28.1. La sécurité dans les spécifications du langage

Les spécifications du langage apportent de nombreuses fonctionnalités pour renforcer la sécurité du code aussi bien lors de la phase de compilation que lors de la phase d'exécution :

- typage fort (toutes les variables doivent posséder un type)
- initialisation des variables d'instances avec des valeurs par défaut
- modificateur d'accès pour gérer l'encapsulation et donc l'accessibilité aux membres d'un objet
- les membres final
- ...

28.1.1. Les contrôles lors de la compilation

28.1.2. Les contrôles lors de l'exécution

La JVM exécute un certain nombre de contrôle au moment de l'exécution :

- vérification des accès en dehors des limites des tableaux
- contrôle de l'utilisation des casts
- vérification par le classloader de l'intégrité des classes utilisées
- ...

28.2. Le contrôle des droits d'une application

Un système de contrôle des droits des applications a été intégré à Java dès sa première version notamment pour permettre de sécuriser l'exécution des applets. Ces applications téléchargées sur le réseau et exécutées sur le poste client doivent impérativement assurer aux personnes qui les utilisent que celles−ci ne risquent pas de réaliser des actions malveillantes sur le système dans lequel elles s'exécutent.

Le modèle de sécurité relatif aux droits des applications développées en Java a évolué au fur et à mesure des différentes versions de Java.

28.2.1. Le modèle de sécurité de Java 1.0

Le modèle proposé par Java 1.0 est très sommaire puisqu'il ne distingue que deux catégories d'applications :

- les applications locales
- les applications téléchargées sur le réseau

Le modèle est basé sur le "tout ou rien". Les applications locales ont tous les droits et les applications téléchargées ont des droits très limités. Les restrictions de ces dernières sont nombreuses :

- impossibilité d'écrire sur le disque local
- impossibilité d'obtenir des informations sur le système local
- impossibilité de se connecter à un autre serveur que celui d'ou l'application a été téléchargée
- ...

La mise en oeuvre de ce modèle est assuré par le "bac à sable" (sand box en anglais) dans lequel s'exécute les applications téléchargées.

28.2.2. Le modèle de sécurité de Java 1.1

Le modèle proposé par la version 1.0 est très efficace mais beaucoup trop restrictif surtout dans le cadre d'utilisation personnelle tel que des applications pour un intranet par exemple.

Le modèle de la version 1.1 propose la possibilité de signer les applications packagées dans un fichier .jar. Une application ainsi signée possède les mêmes droits qu'une application locale.

28.2.3. Le modèle Java 1.2

Le modèle proposé par la version 1.1 a apporté de début de solution pour attribuer des droits sensibles à certaines applications. Mais ce modèle manque cruellement de souplesse puisqu'il s'appuit toujours sur le modèle "tout au rien".

Le modèle de la version 1.2 apporte enfin une solution très souple mais plus compliquée à mettre en oeuvre.

Les droits accordés à une application sont rassemblés dans un fichier externe au code qui se nomme politique de sécurité. Ces fichiers se situe dans le répertoire lib/security du répertoire ou est installé le JRE. Par convention, ces fichiers ont pour extension .policy.

28.3. JCE (Java Cryptography Extension)

JCE est une API qui propose de standardiser l'utilisation de la cryptographie en restant indépendant des algorithmes utilisés. Elle prend en compte le cryptage/décryptage de données, la génération de clés, et l'utilisation de la technolgoie MAC (Message Authentication Code) pour garantir l'intégrité d'un message.

JCE a été intégré au JDK 1.4. Auparavant, cette API était disponible en tant qu'extension pour les JDK 1.2 et 1.3.

Pour pouvoir utiliser cette API, il faut obligatoirement utiliser une implémentation développée par un fournisseur (provider). Avec le JDK 1.4, Sun fournit une implémentation de référence nommée SunJCE.

Les classes et interfaces de l'API sont regroupées dans le package javax.crypto

28.3.1. La classe Cipher

Cette classe encapsule le cryptage et le décyptage de données.

La méthode statique getInstance() permet d'obtenir une instance particulière d'un algorithme fourni par un fournisseur. Le nom de l'algorithme est fourni en paramètre de la méthode sous la forme d'une chaîne de caractères.

Avant la première utilisation de l'instance obtenue, il faut initialiser l'objet en utilisant une des nombreuses surcharges de la méthode init().

28.4. JSSE (Java Secure Sockets Extension)

Les classes et interfaces de cette API sont regroupées dans les packages javax.net et javax.net.ssl.

28.5. JAAS (Java Authentication and Authorization Service)

Les classes et interfaces de cette API sont regroupées dans le package javax.security.auth

Cette API a été intégrée au J.D.K. 1.4.

29. Java Web Start (JWS)

Java Web Start est une technologie pour permettre le déploiement d'application standalone à travers le réseau, développée avec la plate forme Java 2. Il permet l'installation d'une application grâce à un simple clic dans un navigateur. JWS a été inclus dans le J2RE 1.4. Pour les versions antérieures du J2RE, il est nécessaire de télécharger JWS et de l'installer sur le poste client.

JWS est le résultat des travaux de la JSR−56. La page officielle de Sun concernant cette technologie est : http://java.sun.com/products/javawebstart/

JWS permet la mise à jour automatique de l'application si une nouvelle version est disponible sur le serveur et assure une mise en cache locale des applications pour accélérer leur réutilisation ultérieure.

La sécurité des applications exécutées est assurée par l'utilisation du bac à sable (sandbox) comme pour une applet, dès lors pour certaines opérations il est nécessaire de signer l'application.

JWS utilise et implémente une API et un protocole nommée Java Network Lauching Protocol (JPNL).

Le grand avantage de Java Web Start est qu'il est inutile de modifier une application pour qu'elle puisse être déployée avec cette technologie (à condition que les fichiers contenant des ressources soient accédés en utilisant la méthode getResource() du classloader).

L'application doit être packagée dans un fichier jar qui sera associée sur le serveur à un fichier particulier de lancement.

L'utilisation d'une application via JWS implique la réalisation de plusieurs étapes :

- Packager l'application dans un fichier jar en le signant si nécessaire
- Créer le fichier de lancement .jnlp
- Copier les deux fichiers sur le serveur web

Ce chapitre contient plusieurs sections :

29.1. Création du package de l'application

L'application doit être packagée sous la forme d'un fichier .jar.

Il est possible de fournir une petite icône pour représenter l'application : celle si doit avoir une taille de 64 x 64 pixels au format Gif ou JPEG.

29.2. Signer un fichier jar

L'exemple de cette section crée un certificat et signe l'application avec ce dernier.

Exemple :

```
C:\>keytool −genkey −keystore mes_cles −alias cle_de_test
Tapez le mot de passe du Keystore :  test
Mot de passe de Keystore trop court, il doit compter au moins 6 caractères
Tapez le mot de passe du Keystore :  erreur keytool : java.lang.NullPointerExcep
tion
C:\>keytool −genkey −keystore mes_cles −alias cle_de_test
Tapez le mot de passe du Keystore :  mptest
Quels sont vos prénom et nom ?
  [Unknown] :  jean michel
Quel est le nom de votre unité organisationnelle ?
  [Unknown] :  test
Quelle est le nom de votre organisation ?
  [Unknown] :  test
Quel est le nom de votre ville de résidence ?
  [Unknown] :  Metz
Quel est le nom de votre état ou province ?
  [Unknown] :  France
Quel est le code de pays à deux lettres pour cette unité ?
  [Unknown] :  fr
Est−ce CN=jean michel, OU=test, O=test, L=Metz, ST=France, C=fr ?
  [non] :  oui
Spécifiez le mot de passe de la clé pour <cle_de_test>
        (appuyez sur Entrée s'il s'agit du mot de passe du Keystore) :
C:\>
C:\>keytool −selfcert −alias cle_de_test −keystore mes_cles
Tapez le mot de passe du Keystore :  mptest
C:\>keytool −list −keystore mes_cles
Tapez le mot de passe du Keystore :  mptest
Type Keystore : jks
Fournisseur Keystore : SUN
Votre Keystore contient 1 entrÚe(s)
cle_de_test, 12 nov. 2003, keyEntry,
Empreinte du certificat (MD5) : 9E:5A:61:CC:D8:88:02:59:1D:3B:41:C9:CA:26:1D:BD
```

29.3. Le fichier JNPL

Ce fichier au format XML permet de décrire l'application.

La racine de ce document XML est composé du tag <jnpl>. Son attribut codebase permet de préciser l'url ou sont stockés les fichiers précisés dans le document via l'attribut href.

La tag <information> permet de fournir des précisions qui seront utilisées par le gestionnaire d'application sur le poste client. Ce tag possède plusieurs noeud enfants :

Nom du tag	Rôle
Title	Le nom de l'application
Vendor	Nom de l'auteur de l'application
Homepage	Préciser une page HTML qui contient des informations sur l'application grâce à son attribut href
Description	Une description de l'application. Il est possible de préciser plusieurs type description grâce à l'attribut kind. Les valeurs possibles sont : one−line, short et tooltip. Pour utiliser plusieurs descriptions, il faut

	utiliser plusieurs tags Description avec l'attribut kind adéquat
Offline−allowed	Ce tag précise que l'application peut être exécutée dans un mode déconnecté. L'avantage de ne pas préciser ce tag et de s'assurer que la dernière version de l'application est toujours utilisée mais elle nécessite obligatoirement une connexion pour toute exécution.
Icon	Permet de préciser une URL vers une image de 64 x 64 pixels au format gif ou JPEG grâce à l'attribut href

Le tag <security> permet de préciser des informations concernant la sécurité.

Nom du tag	Rôle
All−permissions	Indique que l'application à besoin de tous les droits pour s'exécuter. L'application doit alors être obligatoirement signée. Si ce tag n'est pas précisé alors l'application s'exécute dans le bac à sable et possède les mêmes restrictions qu'une applet au niveau de la sécurité

Le tag <resources> permet de préciser des informations sur les ressources utilisées par l'application. L'attribut os permet de préciser des paramètres pour un système d'exploitation particulier.

Nom du tag	Rôle
J2se	Permet de préciser les JRE qui peuvent être utilisés par l'application. Les valeurs utilisables par l'attribut version sont 1.2, 1.3 et 1.4. Il est possible de préciser un numéro de version particulier ou d'utiliser le caractère * pour préciser n'importe quel numéro de release. L'ordre des différentes valeurs fournies est important.
Jar	Permet de préciser un fichier .jar qui est utilisé par l'application
Nativelib	Permet de préciser une bibliothèque utilisé par l'application qui contient du code natif
Property	Permet de préciser une propriété système qui sera utilisable par l'application. L'attribut name permet de préciser le nom de la propriété et l'attribut value permet de préciser sa valeur

Le tag <application−desc> permet de préciser la classe qui contient la méthode main() grâce à son attribut main−class.

Nom du tag	Rôle
Argument	Permet de préciser des arguments à l'application tel qu'il pourrait être fourni sur une ligne de commande

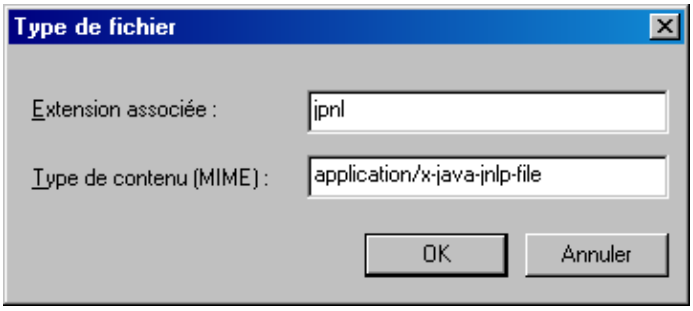
Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost/" href="MonApplication.jnlp">
  <information>
    <title>Mon Application</title>
    <vendor>Jean Michel</vendor>
    <homepage href="http://localhost/"/>
    <description>Mon application</description>
    <description kind="short">une application de test</description>
    <offline-allowed/>
  </information>
  <security>
  </security>
  <resources>
    <j2se version="1.4"/>
    <jar href="MonApplication.jar"/>
  </resources>
  <application-desc main-class="com.moi.dej.jnlp.MonApplication" />
</jnlp>
```

29.4. Configuration du serveur web

Le serveur qui va servir les fichiers doit être configuré pour qu'il associe le type MIME « application/x−java−jnlp−file » avec l'extension .jnpl

Par exemple sous IIS 5, il faut utiliser l'option propriété du menu contextuel du site. Dans l'onglet « En−Tête http », cliquez sur le bouton « Types de fichiers ». Dans le boîte de dialogue « Type de fichiers », cliquez sur le bouton « Nouveau type » si l'association n'est pas présente dans la liste. Une boîte de dialogue permet de saisir l'extension et le type MIME



Le type MIME permet au navigateur de connaître l'application qui devra être utilisée lors de la réception des données du serveur web.

29.5. Fichier HTML

Hormis le code miminum requis par la norme HTML, la seule chose indispensable est un lien dont l'URL pointe vers le fichier .jnpl sur le serveur web.

Exemple :

```
<html>
<head>
<title>Mon Application</title>
</head>
<body>
<H1>Mon Application</H1>
<a href="http://localhost/Monapplication.jnlp">Lancez MonApplication</a>
</body>
</html>
```

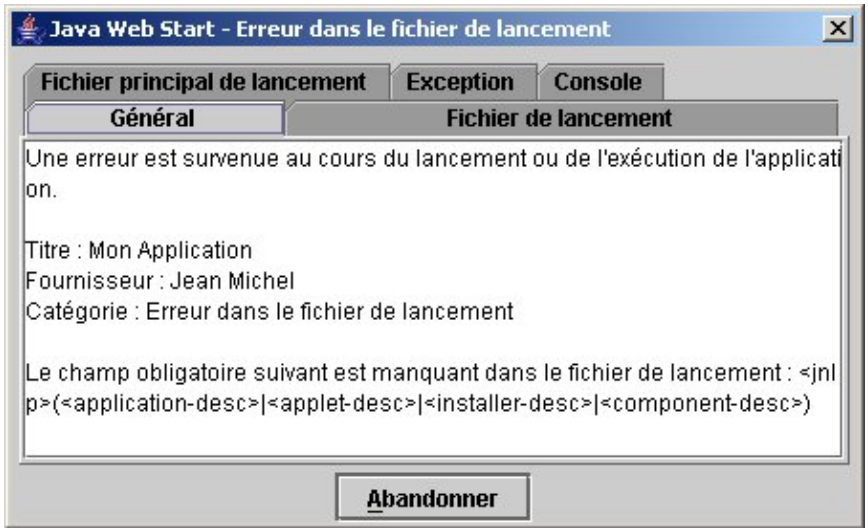
29.6. Tester l'application

Il faut ouvrir un navigateur et saisir l'url de la page contenant le lien vers le fichier jnlp

Java Web Start se lance

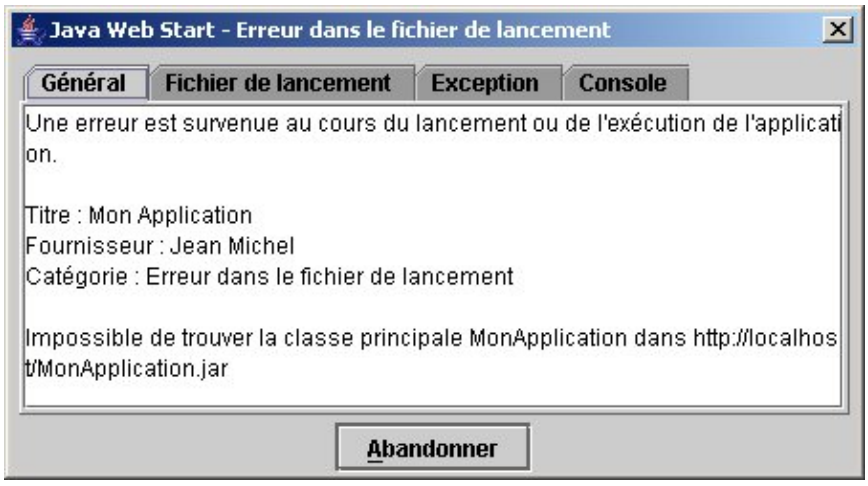
Si le fichier jnlp contient une erreur alors un message d'erreur est affiché.

Cliquez sur « Détails » pour obtenir des informations sur l'erreur.



Si l'application nécessite un accès au système et que le fichier jar n'est pas signé, alors un message erreur est affiché :

Si la classe précisée n'est pas trouvée dans le fichier jar indiqué alors un message d'erreur est affiché



Dans cet exemple, pour résoudre le problème il faut indiquer le nom pleinement qualifié de la classe.

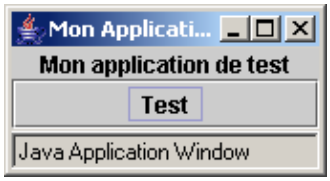
Au premier démarrage réussi d'une application, JWS demande si l'on souhaite créer un raccourci sur le bureau.

En cliquant sur le bouton «Oui», JWS créé un raccourci sur le bureau.

Exemple de raccourci :

```
"C:\Program Files\Java\j2re1.4.2_02\javaws\javaws.exe"
"@C:\Documents and Settings\administrateur\Application Data\
Sun\Java\Deployment\javaws\cache\indirect\indirect31560.ind"
```

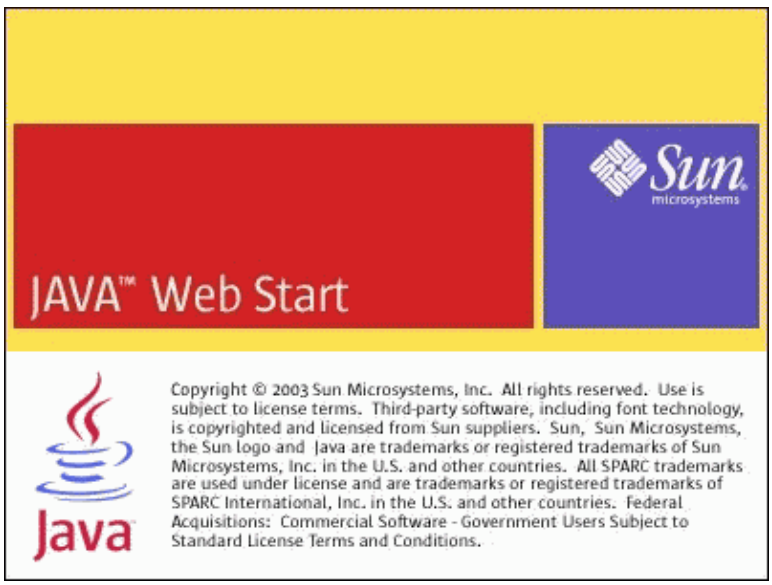
L'application se lance

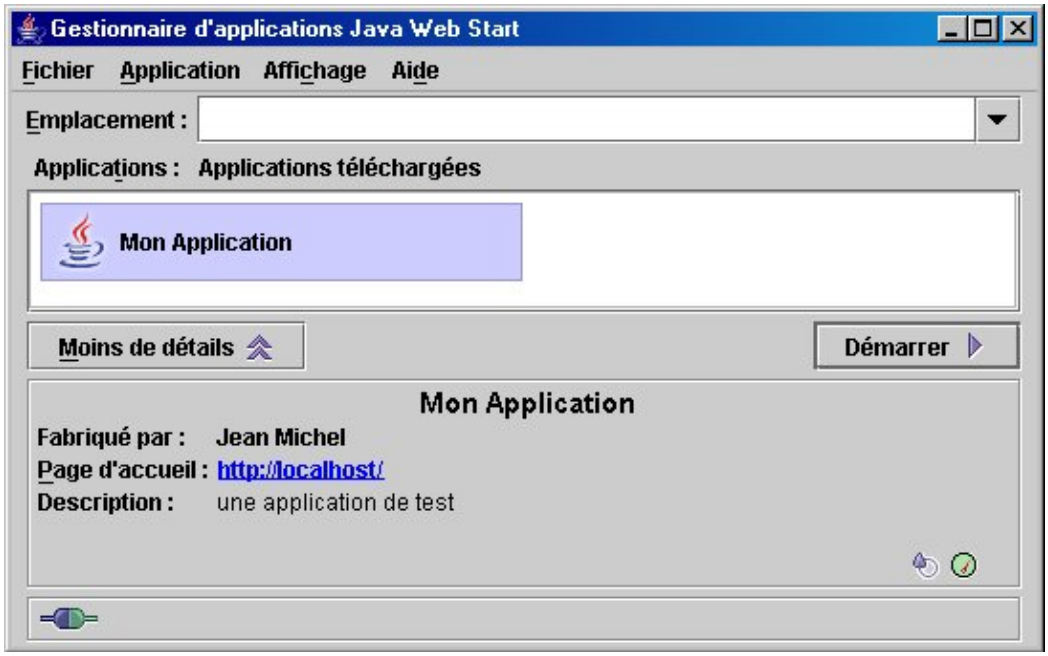


Comme pour les applets, par mesure de sécurité, un petit libellé en bas des fenêtres est affiché indiquant que la fenêtre est issue de l'exécution d'une application Java.

29.7. Utilisation du gestionnaire d'applications

Pour lancer le gestionnaire d'applications, il suffit de double cliquer sur l'icône de « Java Web Start » sur le bureau.





Le gestionnaire d'application permet de gérer les applications en local : il permet de lancer les applications déjà téléchargées sur le poste et les mettre à jour.

Plusieurs petites icônes peuvent apparaître selon le contexte

 : une mise à jour de l'application est téléchargeable sur le serveur

 : l'application peut être exécutée sans connexion au réseau

 : l'application est mise en cache en local

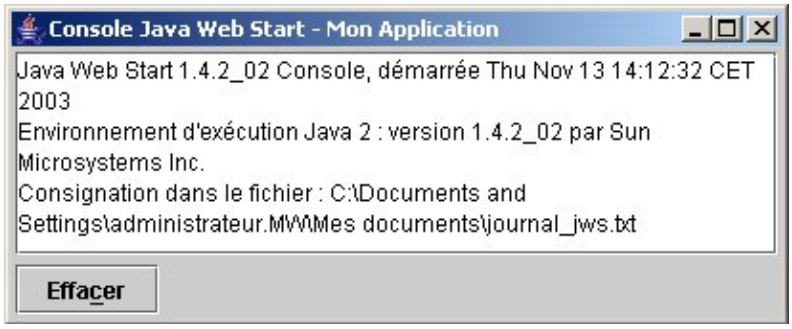
 : l'application n'est pas signée

29.7.1. Lancement d'une application

Pour lancer l'application, il suffit de sélectionner l'application concernée et de cliquer sur le bouton « Démarrer ».

29.7.2. Affichage de la console

Dans les préférences, sur l'onglet « Avancé », cocher la case à cocher « Afficher la console Java »



29.7.3. Consigne dans un fichier de log

Il permet aussi de configurer JWS. Par exemple, en cas de problème, il est possible de demander de consigner une trace d'exécution dans un fichier journal. Celui est particulièrement utile lors du débogage.

Il est possible d'enregistrer les actions dans un fichier de log. Pour cela, il faut cocher la case « Consigner les sorties » et cliquer sur le bouton « Choisir le nom du fichier journal » pour sélectionner ou saisir le nom du fichier.

Exemple :

```
Java Web Start 1.4.2_02 Console, démarrée Thu Nov 13 13:54:36 CET 2003
Environnement d'exécution Java 2 : version 1.4.2_02 par Sun Microsystems Inc.
Consignation dans le fichier : C:\Documents and Settings\admin\Mes documents\journal_jws.txt
Java Web Start 1.4.2_02 Console, démarrée Thu Nov 13 13:54:41 CET 2003
Environnement d'exécution Java 2 : version 1.4.2_02 par Sun Microsystems Inc.
Consignation dans le fichier : C:\Documents and Settings\admin\Mes documents\journal_jws.txt
Java Web Start 1.4.2_02 Console, démarrée Thu Nov 13 13:55:14 CET 2003
Environnement d'exécution Java 2 : version 1.4.2_02 par Sun Microsystems Inc.
Consignation dans le fichier : C:\Documents and Settings\admin\Mes documents\journal_jws.txt
java.lang.NullPointerException
            at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
            at java.lang.ClassLoader.loadClass(Unknown Source)
            at java.lang.ClassLoader.loadClass(Unknown Source)
            at com.sun.javaws.Launcher.continueLaunch(Unknown Source)
            at com.sun.javaws.Launcher.handleApplicationDesc(Unknown Source)
            at com.sun.javaws.Launcher.handleLaunchFile(Unknown Source)
            at com.sun.javaws.Launcher.run(Unknown Source)
            at java.lang.Thread.run(Unknown Source)
```

29.8. L'API de Java Web Start

La suite de ce chapitre sera développée dans une version future de ce document

30. JNI (Java Native Interface)

JNI est l'acronyme de Java Native Interface. C'est une technologie qui permet d'utiliser du code natif dans une classe Java notamment C.

L'inconvénient majeur de cette technologie est d'annuler la portabilité du code Java. En contre partie cette technologie peut être très utile dans plusieurs cas :

- pour des raisons de performance
- utiliser des composants éprouvés déjà existants

La mise en oeuvre de JNI nécessite plusieurs étapes :

- la déclaration et l'utilisation de la ou des méthodes natives dans la classe Java
- la compilation de la classe Java
- la génération du fichier d'en−tête avec l'outil javah
- l'écriture du code natif en utilisant entre autre les fichiers d'en−tête fourni par le JDK et celui généré précédemment
- la compilation du code natif sous la forme d'une bibliothèque

Le format de la bibliothèque est donc dépendante du système d'exploitation pour lequel elle est développée : .dll pour les systèmes de type Windows, .so pour les système de type Unix, ...

Ce chapitre contient plusieurs sections :

30.1. Déclaration et utilisation d'une méthode native

La déclaration dans le code source Java est très facile puisqu'il suffit de déclarer la signature de la méthode avec le modificateur native. Le modificateur permet au compilateur de savoir que cette méthode est contenue dans une bibliothèque native.

Il ne doit pas y avoir d'implémentation même un corps vide pour une méthode déclarée native.

Exemple :

```
class TestJNI1 {
public native void afficherBonjour();
static {
System.loadLibrary("mabibjni");
    }

public static void main(String[] args) {
new TestJNI1().afficherBonjour();
```

```
    }
}
```

Pour pouvoir utiliser une méthode native, il faut tout d'abord charger la bibliothèque. Pour réaliser ce chargement, il utiliser la méthode statique loadLibrary() de la classe system. Il faut obligatoirement s'assurer que la bibliothèque est chargée avant le premier appel de la méthode native.

Le plus simple pour assurer ce chargement est de le demander dans un morceau de code d'initialisation statique de la classe.

Exemple :

```
class TestJNI1 {
  public native void afficherBonjour();
  static {
    System.loadLibrary("mabibjni");
  }
}
```

Le nom de la bibliothèque fournie en paramètre doit être indépendant de la plate−forme utilisée : il faut préciser le nom de la bibliothèque sans son extension. Le nom sera automatiquement adapté selon le système d'exploitation sur lequel le code Java est exécuté.

L'utilisation de la méthode native dans le code Java se fait de la même façon qu'une méthode classique.

Exemple :

```
class TestJNI1 {
  public native void afficherBonjour();
  static {
    System.loadLibrary("mabibjni");
  }

  public static void main(String[] args) {
    new TestJNI1().afficherBonjour();
  }
}
```

30.2. La génération du fichier d'en−tête

L'outil javah fourni avec le JDK permet de générer un fichier d'en−tête qui va contenir la définition dans le langage C des fonctions correspondant aux méthodes déclarées native dans le source Java.

Javah utilise le byte code pour générer le fichier .h. Il faut donc que la classe Java soit préalablement compilée.

La syntaxe est donc : javah –jni nom_fichier_sans_extension

Exemple :

```
D:\java\test\jni>dir
03/12/2003  14:39       <DIR>          .
03/12/2003  14:39       <DIR>          ..
03/12/2003  14:39                   230 TestJNI1.java
            2 fichier(s)              230 octets
            2 Rép(s)   2 200 772 608 octets libres
D:\java\test\jni>javac TestJNI1.java
D:\java\test\jni>javah −jni TestJNI1
D:\java\test\jni>dir
Répertoire de D:\java\test\jni
03/12/2003  14:39       <DIR> .
```

```
03/12/2003  14:39       <DIR> ..
03/12/2003  14:39                 459 TestJNI1.class
03/12/2003  14:39                 399 TestJNI1.h
03/12/2003  14:39                 230 TestJNI1.java
            3 fichier(s) 1 088 octets
            2 Rép(s) 2 198 208 512 octets libres
D:\java\test\jni>
```

Le fichier TestJNI1.h généré est le suivant :

Exemple :

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include
/* Header for class TestJNI1 */

#ifndef _Included_TestJNI1
#define _Included_TestJNI1
#ifdef __cplusplus
extern " C " {
#endif
/*
 *      Class:     TestJNI1
 *      Method:    afficherBonjour
 *      Signature: ()V
 */
JNIEXPORT void JNICALL Java_TestJNI1_afficherBonjour
(JNIEnv *, jobject);
#ifdef __cplusplus
}
#endif
#endif
```

Le nom de chaque fonction native respecte le format suivant :
Java_nomPleinementQualifieDelaClasse_NomDeLaMethode

Ce fichier doit être utilisé dans l'implémentation du code de la fontion.

Même si la méthode native est déclarée sans paramètre, il y a toujours deux paramètres passés à la fonction native :

- un pointeur vers un structure JniEnv : cet objet permet d'utiliser certaines fonctions permettant d'utiliser certains paramètres non primitifs fournis à la méthode
- jobject qui est l'objet lui même : c'est l'équivalent du mot clé this dans le code Java

30.3. L'écriture du code natif en C

La bibliothèque contenant la ou les fonctions qui seront appelées doit être écrite dans un langage (c ou c++) et compilée.

Pour l'écriture en C, facilitée par la génération du fichier.h, il est nécessaire en plus des includes liées au code des fonctions d'inclure deux fichiers d'en-tête :

- jni.h qui est fourni avec le JDK
- le fichier .h généré par la commande javah

Exemple : TestJNI.c

```
#include <jni.h>
#include <stdio.h>
#include “TestJNI1.h”

JNIEXPORT void JNICALL
Java_TestJNI1_afficherBonjour(JNIEnv *env, jobject obj)
```

```
{
  printf(« Bonjour\n »);
  return;
}
```

Il faut compiler ce fichier source sous la forme d'un fichier objet .o

Exemple : avec MinGW sous Windows

```
D:\java\test\jni>gcc −c −I”C:\j2sdk1.4.2_02\include” −I”C:\j2sdk1.4.2_02\include
\win32” −o TestJNI.o TestJNI.c
```

Il faut ensuite définir un fichier .def qui contient la définition des fonctions exportées par la bibliothèque

Exemple : TestJNI.def

```
EXPORTS
Java_TestJNI1_afficherBonjour
```

Il ne reste plus qu'à générer la dll.

Exemple : TestJNI.def

```
D:\java\test\jni>gcc −shared −o mabibjni.dll TestJNI.c TestJNI.def
Warning: resolving _Java_TestJNI1_afficherBonjour by linking to _Java_TestJNI1_a
fficherBonjour@8
Use−enable−stdcall−fixup to disable these warnings
Use−disable−stdcall−fixup to disable these fixups

D:\java\test\jni>dir
Répertoire de D:\java\test\jni
03/12/2003  16:22       <DIR>          .
03/12/2003  16:22       <DIR>          ..
03/12/2003  16:22               12 017 mabibjni.dll
03/12/2003  15:58                  193 TestJNI.c
03/12/2003  16:20                   40 TestJNI.def
03/12/2003  16:04                  543 TestJNI.o
03/12/2003  14:39                  459 TestJNI1.class
03/12/2003  14:39                  399 TestJNI1.h
03/12/2003  14:39                  230 TestJNI1.java
               9 fichier(s)          14 074 octets
               2 Rép(s)   2 198 392 832 octets libres

D:\java\test\jni>
```

Il ne reste plus qu'à exécuter le code Java dans une machine virtuelle.

Exemple :

```
D:\java\test\jni>java TestJNI1
Bonjour
D:\java\test\jni>
```

Il est intéressant de noter que tant que la signature de la méthode native ne change pas, il est inutile de recompiler la classe Java si l& fonction dans la bibliothèque est modifiée et recompilée.

30.4. Passage de paramètres et renvoi d'une valeur (type primitif)

Une méthode a quasiment toujours besoin de paramètres et souvent besoin de retourner une valeur.

Cette section va définir et utiliser une méthode native qui ajoute deux entiers et renvoie le résultat de l'addition.

Exemple : le code Java

```
class TestJNI1 {
  public native int ajouter(int a, int b);
    static {
      System.loadLibrary("mabibjni");
    }

  public static void main(String[] args) {
    TestJNI1 maclasse = new TestJNI1();
    System.out.println("2 + 3 = " + maclasse.ajouter(2,3));
  }
}
```

La déclaration de la méthode n'a rien de particulier hormis le modificateur native.

La signature de la fonction dans le fichier .h tient des paramètres.

Exemple :

```
JNIEXPORT jint JNICALL Java_TestJNI1_ajouter
(JNIEnv *, jobject, jint, jint);
```

Les deux paramètres sont ajoutés dans la signature de la fonction avec un type particulier jint, défini avec un typedef dans le fichier jni.h. Il y a d'ailleurs des définitions pour toutes les primitives.

Primitive Java	Type natif
boolean	jboolean
byte	jbyte
char	jchar
double	jdouble
int	jint
float	jfloat
long	jlong
short	jshort
void	void

Il suffit ensuite d'écrire l'implémentation du code natif.

Exemple :

```
#include <jni.h>
#include <stdio.h>
#include "TestJNI2.h"

JNIEXPORT jint JNICALL Java_TestJNI2_ajouter
(JNIEnv *env, jobject obj, jint a, jint b)
{
  return a + b;
}
```

Il faut ensuite compiler le code :

Exemple :

```
D:\java\test\jni>gcc −c −I"C:\j2sdk1.4.2_02\include" −I"C:\j2sdk1.4.2_02\include
\win32" −o TestJNI2.o TestJNI2.c
```

Il faut définir le fichier .def : l'exemple ci dessous va construire une bibliothèque qui va contenir les fonctions natives des deux classes Java précédemment définies.

Exemple :

```
EXPORTS
Java_TestJNI1_afficherBonjour
Java_TestJNI2_ajouter
```

Il suffit de générer la bibilothèque.

Exemple :

```
D:\java\test\jni>gcc −shared −o mabibjni.dll TestJNI.c TestJNI2.c TestJNI.def
Warning: resolving _Java_TestJNI1_afficherBonjour by linking to _Java_TestJNI1_a
fficherBonjour@8
Use−enable−stdcall−fixup to disable these warnings
Use−disable−stdcall−fixup to disable these fixups
Warning: resolving _Java_TestJNI2_ajouter by linking to _Java_TestJNI2_ajouter@1
6
```

Il ne reste plus qu'a exécuter le code Java

Exemple :

```
D:\java\test\jni>java TestJNI2
2 + 3 = 5
```

30.5. Passage de paramètres et renvoi d'une valeur (type objet)

Les objets sont passés par référence en utilisant une variable de type jobject. Plusieurs autres type sont prédéfinis par JNI pour des objets fréquemment utilisés :

Objet C	Objet Java
jobject	java.lang.object
jstring	java.lang.String
jclass	java.lang.Class
jthrowable	java.lang.Throwable
jarray	type de base pour les tableaux
jintArray	int[]
jlongArray	long[]
jfloatArray	float[]
jdoubleArray	double[]

jobjectArray	Object[]
jbooleanArray	boolean[]
jbyteArray	byte[]
jcharArray	char[]
jshortArray	short[]

Exemple : concaténation de deux chaînes de caractères

```
class TestJNI3 {
  public native String concat(String a, String b);

  static {
    System.loadLibrary("mabibjni");
  }

  public static void main(String[] args) {
    TestJNI3 maclasse = new TestJNI3();
    System.out.println("abc + cde = " + maclasse.concat("abc","cde"));
  }
}
```

La déclaration de la fonction native dans le fichier TestJNI3.h est la suivante :

Exemple :

```
/*
 * Class:     TestJNI3
 * Method:    concat
 * Signature: (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_TestJNI3_concat
  (JNIEnv *, jobject, jstring, jstring);
```

Pour utiliser les paramètres de type jstring dans le code natif, il faut les transformer en utilisant des fonctions proposées par l'interface de type JNIEnv car le type String de Java n'est pas directement compatible avec les chaînes de caractères C (char *). Il existe des fonctions pour transformer des chaînes codées en UTF−8 ou en Unicode.

Les méthodes pour traiter les chaînes au format UTF−8 sont :

- la méthode GetStringUTFChars() permet de convertir une chaîne de caractères Java en une chaîne de caractères de type C.
- la méthode NewStringUTF() permet de demander la création d'une nouvelle chaîne de caractères.
- la méthode GetStringUTFLength() permet de connaître la taille de la chaîne de caractères.
- la méthode ReleaseStringUTFChars() permet de demander la libération des ressources allouées pour la chaîne de caractères dès que celle ci n'est plus utilisée. Son utilisation permet d'éviter des fuites mémoire.

Les méthodes équivalentes pour les chaines de caractères au format unicode sont : GetStringChars(), NewString(), GetStringUTFLength() et ReleaseStringChars()

Exemple : TestJNI3.c

```
#include <jni.h>
#include <stdio.h>
#include "TestJNI3.h"
JNIEXPORT jstring JNICALL Java_TestJNI3_concat
  (JNIEnv *env, jobject obj, jstring chaine1, jstring chaine2){
    char resultat[256];
    const char *str1 = (*env)->GetStringUTFChars(env, chaine1, 0);
    const char *str2 = (*env)->GetStringUTFChars(env, chaine2, 0);
    sprintf(resultat,"%s%s", str1, str2);
    (*env)->ReleaseStringUTFChars(env, chaine1, str1);
```

```
    (*env)->ReleaseStringUTFChars(env, chaine2, str2);
    return (*env)->NewStringUTF(env, resultat);
}
```

Attention : ce code est très simpliste car il ne vérifie pas un éventuel débordement du tableau résultat.

Après la compilation des différents éléments, l'exécution affiche le résultat escompté.

Exemple :

```
D:\java\test\jni>java TestJNI3
abc + cde = abccde
```

31. JDO (Java Data Object)

JDO (Java Data Object) est la spécification du JCP n° 12 qui propose une technologie pour assurer la persistance d'objets Java dans un système de gestion de données. La spécification regroupe un ensemble d'interfaces et de règles qui doivent être implémentées par un fournisseur tiers.

La version 1.0 de cette spécification a été validée au premier trimestre 2002. Elle devrait connaître un grand succès car le mapping entre des données stockées dans un format particulier (bases de données ...) et un objet a toujours été difficile. JDO propose de faciliter cette tâche en fournissant un standard.

Ce chapitre contient plusieurs sections :

31.1. Présentation

Les principaux buts de JDO sont :

- la facilité d'utilisation (gestion automatique du mapping des données)
- la persistance universelle : persistance vers tout type de système de gestion de ressources (bases de données relationnelles, fichiers, ...)
- la transparence vis à vis du système de gestion de ressources utilisé : ce n'est plus le développeur mais JDO qui dialogue avec le système de gestion de ressources
- la standardisation des accès aux données
- la prise en compte des transactions

Le développement avec JDO se déroule en plusieurs étapes :

1. écriture des objets contenant les données (des beans qui encapsulent les données) : un tel objet est nommé instance JDO
2. écriture des objets qui utilisent les objets métiers pour répondre aux besoins fonctionnels. Ces objets utilisent l'API JDO.
3. écriture du fichier metadata qui précise le mapping entre les objets et le système de gestion des ressources. Cette partie est très dépendante du système de gestion de ressources utilisé
4. enrichissement des objets métiers
5. configuration du système de gestion des ressources

JDBC et JDO ont les différences suivantes :

JDBC	JDO
orienté SQL	orienté objets

le code doit être ajouté explicitement	code est ajouté automatiquement
	gestion d'un cache
	mapping réalisé automatiquement ou à l'aide d'un fichier de configuration au format XML
utilisation avec un SGBD uniquement	utilisation de tout type de format de stockage

JDO est une spécification qui définit un standard : pour pouvoir l'utiliser il faut utiliser une implémentation fournie par un fournisseur. Plusieurs implémentations existent et le choix de l'une d'elle doit tenir compte des performances, du prix, du support des cibles de stockage des données, etc ... L'intérêt des spécifications est qu'il est possible d'utiliser le même code avec des implémentations différentes tant que l'on utilise uniquement les fonctionnalités précisées dans les spécifications.

Chaque implémentation est capable d'utiliser un ou plusieurs systèmes de stockage de données particulier (base de données relationnel, base de données objets, fichiers, ...).

Attention : tous les objets ne peuvent pas être rendu persistant avec JDO.

31.2. Un exemple avec Lido

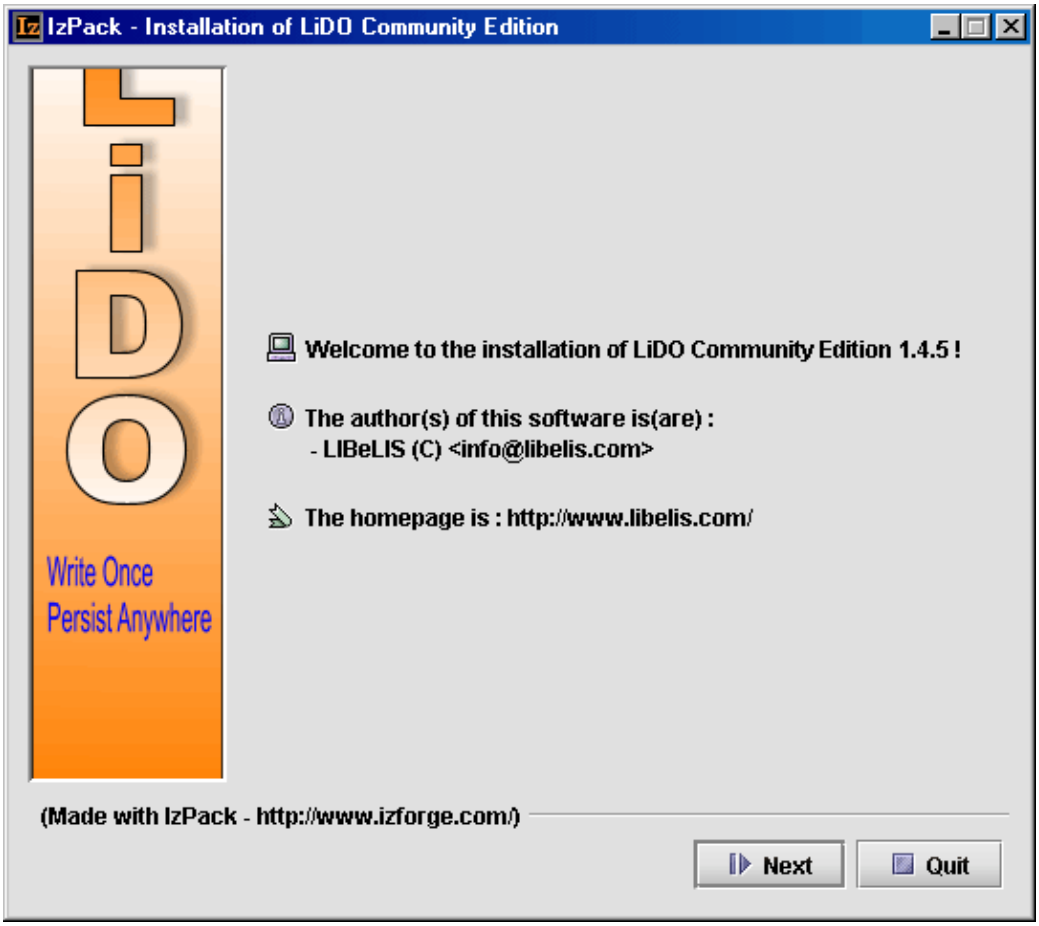
Les exemples de cette section ont été réalisés avec Lido Community Edition version 1.4.5. de la société Libelis.

Cette version est librement téléchargeable après enregistrement à l'url : http://www.libelis.com

Pour lancer l'installation, il suffit de double cliquer sur le fichier LiDO_Community_1[1].4.5.jar ou de saisir la commande :

Installation de Lido community edition de Libelis

```
java –jar LiDO_Community_1[1].4.5.jar
```



L'installation s'opère simplement en suivant les diffentes étapes de l'assistant.

Le premier exemple permet simplement de rendre persistant un objet instancié dans une base de données MySQL.

Pour faciliter la mise en oeuvre des différentes étapes, un script batch pour Windows sera écrit tout au long de cette section et exécuté. Ce script débute par une initialisation de certaines variables d'environnement.

Début du script

```
@echo off
REM - script permettant la compilation, l'enrichissement, la creation du schema de
REM - base de données et l'execution du code de test de JDO avec Lido 1.4.5.
set LIDO_HOME=C:\java\LiDO
set JAVA_HOME=C:\java\j2sdk1.4.2_02

echo initialisation
echo.
SET PATH=%LIDO_HOME%\bin;%JAVA_HOME%\bin

SET CLASSPATH=.;.\mm.mysql-2.0.14-bin.jar
SET CLASSPATH=%CLASSPATH%;%JAVA_HOME%\lib\tools.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido-api.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\jdo_1_0_0.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\j2ee.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\bin
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido-dev.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido-rdb.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido-rt.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido.tasks
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido.tld
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\skinlf.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\connector_1_0_0.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\jta_1_0_1.jar
```

Le début du script initialise 4 variables :

- LIDO_HOME : cette variable contient le chemin du répertoire dans lequel Lido a été installé
- JAVA_HOME : cette variable contient le chemin du répertoire dans lequel J2SDK a été installé
- PATH : cette variable contient les différents répertoires contenant des exécutables
- CLASSPATH : cette variable contient les différents répertoires et fichiers jar necéssaires pour la compilation et l'exécution

Pour des raisons de facilité, le répertoire courant "." est ajouté dans le CLASSPATH. Le pilote JDBC pour MySQL est aussi ajouté à cette variable.

31.2.1. La création de la classe qui va encapsuler les données

Le code de cet objet reste très simple puisque c'est simplement un bean encapsulant une personne contenant des attributs nom, prenom et datenaiss.

Exemple :

```
package testjdo;

import java.util.*;

public class Personne {
      private String nom ="";
      private String prenom = "";
      private Date datenaiss = null;

      public Personne(String pNom, String pPrenom,Date pDatenaiss) {
            nom=pNom;
            prenom=pPrenom;
            datenaiss=pDatenaiss;
      }

      public String getNom() { return nom; }

      public String getPrenom() { return prenom;}

      public Date getDatenaisse() { return datenaiss; }

      public void setNom(String pNom) { nom = pNom; }

      public void setPrenom(String pPrenom) { nom = pPrenom; }

      public void setDatenaiss(Date pDatenaiss) { datenaiss = pDatenaiss; }

}
```

31.2.2. La création de l'objet qui va assurer les actions sur les données

Cet objet va utiliser des objets JDO pour réaliser les actions sur les données. Dans l'exemple ci dessous, une seule action est codée : l'enregistrement dans la table des données du nouvel objet de type Personne instancié.

Exemple :

```
package testjdo;

import javax.jdo.*;
import java.util.*;

public class PersonnePersist {

      private PersistenceManagerFactory pmf = null;
      private PersistenceManager        pm = null;
      private Transaction               tx  = null;
```

```
      public PersonnePersist() {
            try {

            pmf = (PersistenceManagerFactory) (
            Class.forName("com.libelis.lido.PersistenceManagerFactory").newInstance());
            pmf.setConnectionDriverName("org.gjt.mm.mysql.Driver");
            pmf.setConnectionURL("jdbc:mysql://localhost/testjdo");
      } catch (Exception e ){
            e.printStackTrace();
            }
      }

      public void enregistrer() {
            Personne p = new Personne("mon nom","mon prenom",new Date());
            pm = pmf.getPersistenceManager();
            tx = pm.currentTransaction();
            tx.begin();
            pm.makePersistent(p);
            tx.commit();
            pm.close();
      }

      public static void main(String args[]) {
            PersonnePersist pp = new PersonnePersist();
            pp.enregistrer();
      }
}
```

31.2.3. La compilation

Les deux classes définies ci dessus doivent être compilées normalement en utilisant l'outil javac.

La suite du script : compilation des classes

```
...
echo Compilation en cours
javac −classpath %CLASSPATH% testjdo\*.java
echo Compilation effectuee
echo.
...
```

31.2.4. La définition d'un fichier metadata

Le fichier metadata est un fichier au format XML qui précise le mapping à réaliser.

Exemple : metadata.jdo

```
<? xml version="1.0" ?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
  <package name="testjdo">
    <class name="Personne" identity−type="datastore">
      <field name="nom" />
      <field name="prenom" />
      <field name="datenaiss" />
    </class>
  </package>
</jdo>
```

31.2.5. L'enrichissement des classes contenant des données

Pour assurer une bonne exécution, il faut enrichir l'objet Personne compilé avec du code pour assurer la persistance par JDO. Lido fourni un outil pour réaliser cette tâche. Cet outil est complètement dépend de l'implementation qui en est faite par le fournisseur de la solution JDO.

La suite du script : enrichissement

```
...
echo Enrichissement
java −cp %CLASSPATH% com.libelis.lido.Enhance −metadata metadata.jdo −verbose
echo Enrichissement effectue
echo.
...
```

Le fichier Personne.class est enrichi (sa taille passe de 867 octets à 9693 octets)

31.2.6. La définition du schéma de la base de données

Lido fournit un outil qui permet de générer les tables de la base de données contenant les tables pour le mapping des données plus des tables "techniques" nécessaires aux traitements.

Les paramètres nécessaires à l'outil de Libelis pour définir le schéma de la base de données doivent être rassemblés dans un fichier .properties.

Exemple : propriété pour une base de données de type MySQL

```
# lido.properties file
# jdo standard properties
javax.jdo.option.connectionURL=jdbc:mysql://localhost/testjdo
javax.jdo.option.ConnectionDriverName=org.gjt.mm.mysql.Driver
javax.jdo.option.connectionUserName=root
javax.jdo.option.connectionPassword=
javax.jdo.option.msWait=5
javax.jdo.option.multithreaded=false
javax.jdo.option.optimistic=false
javax.jdo.option.retainValues=false
javax.jdo.option.restoreValues=true
javax.jdo.option.nontransactionalRead=true
javax.jdo.option.nontransactionalWrite=false
javax.jdo.option.ignoreCache=false


# set to PM, CACHE, or SQL to have some traces
# ex:
#lido.trace=SQL,DUMP,CACHE

#  set the Statement pool size
lido.sql.poolsize=10
lido.cache.entry−type=weak

# set the max batched statement
# 0: no batch
# default is 20

lido.sql.maxbatch=30
lido.objectpool=90

# set for PersistenceManagerFactory pool limits
lido.minPool=1
lido.maxPool=10

jdo.metadata=metadata.jdo
```

Il suffit alors d'utiliser l'application DefineSchema fournie par Lido en lui passant en paramètre le fichier .properties et le fichier .jdo

La suite du script : création du schéma de la base de données

```
...
echo DefineSchema en cours
java com.libelis.lido.DefineSchema −properties testjdo.properties −metadata metadata.jdo
echo DefineSchema termine
echo.
...
```

Il est facile de vérifier les traitements effectués par l'outil DefineSchema :

Exemple :

```
C:\java\testjdo>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25 to server version: 4.0.16−nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use testjdo
Database changed
mysql> show tables;
+−−−−−−−−−−−−−−−−−−+
| Tables_in_testjdo |
+−−−−−−−−−−−−−−−−−−+
| lidoidmax         |
| lidoidtable       |
| t_personne        |
+−−−−−−−−−−−−−−−−−−+
3 rows in set (0.00 sec)

mysql> describe lidoidmax;
+−−−−−−−−−−+−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
| Field    | Type       | Null | Key | Default | Extra |
+−−−−−−−−−−+−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
| LIDOLAST | bigint(20) | YES  |     | NULL    |       |
+−−−−−−−−−−+−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
1 row in set (0.00 sec)

mysql> describe lidoidtable;
+−−−−−−−−−−+−−−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
| Field    | Type         | Null | Key | Default | Extra |
+−−−−−−−−−−+−−−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
| LIDOID   | bigint(20)   |      | PRI | 0       |       |
| LIDOTYPE | varchar(255) | YES  | MUL | NULL    |       |
+−−−−−−−−−−+−−−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
2 rows in set (0.00 sec)

mysql> describe t_personne;
+−−−−−−−−−−−+−−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
| Field     | Type        | Null | Key | Default | Extra |
+−−−−−−−−−−−+−−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
| LIDOID    | bigint(20)  |      | PRI | 0       |       |
| nom       | varchar(50) | YES  |     | NULL    |       |
| prenom    | varchar(50) | YES  |     | NULL    |       |
| datenaiss | datetime    | YES  |     | NULL    |       |
+−−−−−−−−−−−+−−−−−−−−−−−−−+−−−−−−+−−−−−+−−−−−−−−−+−−−−−−−+
4 rows in set (0.00 sec)

mysql> select * from t_personne;
Empty set (0.39 sec)
```

31.2.7. L'exécution de l'exemple

Exemple :

```
@echo off
REM − script permettant la compilation, l'enrichissement, la creation du schema de
REM − base de données et l'execution du code de test de JDO avec Libelis 1.4.5.
set LIDO_HOME=C:\java\LiDO
set JAVA_HOME=C:\java\j2sdk1.4.2_02

echo initialisation
echo.
SET PATH=%LIDO_HOME%\bin;%JAVA_HOME%\bin

SET CLASSPATH=.;.\mm.mysql−2.0.14−bin.jar
SET CLASSPATH=%CLASSPATH%;%JAVA_HOME%\lib\tools.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido−api.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\jdo_1_0_0.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\j2ee.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\bin
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido−dev.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido−rdb.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido−rt.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido.tasks
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\lido.tld
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\skinlf.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\connector_1_0_0.jar
SET CLASSPATH=%CLASSPATH%;%LIDO_HOME%\lib\jta_1_0_1.jar

echo Compilation en cours
javac −classpath %CLASSPATH% testjdo\*.java
echo Compilation effectuee
echo.

echo Enrichissement
java −cp %CLASSPATH% com.libelis.lido.Enhance −metadata metadata.jdo −verbose
echo Enrichissement effectue
echo.

echo DefineSchema en cours
java com.libelis.lido.DefineSchema −properties testjdo.properties −metadata metadata.jdo
echo DefineSchema termine
echo.

echo Execution du test
java −cp %CLASSPATH% testjdo.PersonnePersist
echo Execution terminee
echo.
```

A l'issu de l'exécution, un enregistrement est créé dans la table qui mappe l'objet Personne.

Exemple :

```
mysql> select * from t_personne;
+−−−−−−−−+−−−−−−−−−+−−−−−−−−−−−−+−−−−−−−−−−−−−−−−−−−−+
| LIDOID | nom     | prenom     | datenaiss           |
+−−−−−−−−+−−−−−−−−−+−−−−−−−−−−−−+−−−−−−−−−−−−−−−−−−−−+
|      1 | mon nom | mon prenom | 2004−01−23 20:06:11 |
+−−−−−−−−+−−−−−−−−−+−−−−−−−−−−−−+−−−−−−−−−−−−−−−−−−−−+
1 row in set (0.00 sec)
```

31.3. L'API JDO

L'API de JDO se compose de deux packages :

- javax.jdo : ce package est à utiliser par les développeurs pour utiliser JDO
- javax.jdo.spi : ce package est à utiliser par les tiers pour développer une implémentation de JDO

Le package javax.jdo contient essentiellement des interfaces ainsi que quelques classes notamment la classe JDOHelper et les diverses exceptions utilisées par JDO. Les interfaces définies sont : Extent, PersistenceManager, PersistenceManagerFactory, Query etTransaction.

Les exceptions définies par l'API JDO sont : JDOCanRetryException, JDODataStoreException, JDOException, JDOFatalDataStoreException, JDOFatalException, JDOFatalInternalException, JDOFatalUserException, JDOUnsupportedOptionException et JDOUserException.

31.3.1. L'interface PersistenceManager

Cette interface définit les méthodes pour l'objet principal de l'API JDO pour les développeurs.

Certaines méthodes permettent de gérer le cycle de vie d'une instance d'un objet de type PersistenceCapable.

void close()	fermer
Transaction currentTransaction()	renvoie la transaction courante
void deletePersistent()	permet de détruire dans la source de données l'instance encapsulée
Extent getExtent(Class, boolean)	renvoie une collection d'instance encapsulant les données dans la source de données
void makePersistent()	permet de rendre persistante les données encapsulées dans l'instance en créant une nouvelle occurence dans le système de gestion de ressources
void evict()	permet de préciser que l'instance n'est plus utilisée
Query newQuery()	renvoie un objet de type Query qui permet d'effectuer des sélections dans la source de données
void refresh()	permet de redonner à une instance les valeurs contenues dans le système de gestion de ressources

Cette interface propose aussi deux méthodes possédant de nombreuses surcharges de la méthode newQuery() pour obtenir une instance d'un objet de type Query.

31.3.2. L'interface PersistenceManagerFactory

Un objet qui implémente cette interface à pour but de fournir une instance d'une classe qui implémente l'interface PersistenceManager. Un tel objet doit être configuré via des propriétés pour instancier un objet de type PersistenceManager. Ces propriétés doivent être fournies à la fabrique avant l'instanciation du premier objet de type PersistenceManager. Il n'est dès lors plus possible de changer la configuration de la fabrique.

Cette interface possède une méthode nommée getPersistenceManager() qui permet d'obtenir une instance de la classe PersistenceManager.

31.3.3. L'interface PersistenceCapable

Cette interface doit être implémentée lors de son enrichissement par la classe qui va contenir des données. Cette classe avant son enrichissement ne doit pas implémenter cette interface : c'est lors de cette phase d'enrichissement que la classe implémentera cette interface et que seront définies les méthodes déclarées par cette interface nécessaires à la persistence

des données. Cet enrichissement peut se faire de deux façons :

- manuellement : ce qui peut être long et fastidieux
- automatiquement avec un outil fourni avec l'implémentation de JDO : généralement cet outil utilise un fichier au format XML pour obtenir les informations nécessaires à la génération des méthodes

Les méthodes définies dans cette interface sont à l'usage de JDO : une fois la classe enrichie, il ne faut surtout pas appeller directement ces méthodes : elles sont toutes préfixées par jdo.

Une classe qui implémente l'interface PersistenceCapable est nommé instance JDO.

31.3.4. L'interface Query

Cette interface définit des méthodes qui permettent d'obtenir des instances représentant des données issues de la source de données.

L'interface définit plusieurs surcharges de la méthode execute() pour exécuter la requête et renvoyer un ensemble d'instances.

La méthode compile() permet de vérifier la requête et préparer son exécution.

La méthode setFilter() permet de préciser un filtre pour la requête.

Une instance d'un objet implémentant l'interface Query est obtenue en utilisant une des nombreuses surcharges de la méthode newQuery() d'un objet de type PersistenceManager.

31.3.5. L'interface Transaction

Cette interface définit les méthodes pour la gestion des transactions avec JDO.

Elle possède trois méthodes principales qui sont classiques dans la gestion des transactions :

- begin : indique le début d'une transaction
- commit : valide la transaction
- roolback : invalide la transaction et annule toutes les opérations qu'elle contient

31.3.6. L'interface Extent

Une classe qui implémente cette interface permet d'encapsuler toute une collection contenant tous les objets d'un type PersistenceCapable particulier. La méthode iterator() renvoie un objet de type Iterator qui permet de parcourir l'ensemble des éléments de la collection.

L'interface Extent ne prévoit actuellement aucun moyen de filter les éléments de la collection et il est uniquement possible d'obtenir toutes les occurences.

La méthode close(Iterator) permet de fermer l'objet de type Iterator passé en paramètre.

31.3.7. La classe JDOHelper

La classe JDOHelper permet de faciliter l'utilisation de JDO grâce à plusieurs méthodes statiques pouvant être regroupées dans plusieurs catégories :

- connaitre l'état d'une instance JDO.

Nom	Rôle
boolean isDeleted(Object)	renvoie un booléen qui précise si l'instance JDO fournie en paramètre vient d'être supprimée dans le système de gestion de ressources
boolean isDirty(Object)	renvoie un booléen qui précise si l'instance JDO a été modifiée dans la transaction courante
boolean isNew(Object)	renvoie un booléen qui précise si l'instance JDO fournie en paramètre vient d'être rendue persistante en créant une nouvelle instance dans le système de gestion de ressources
boolean isPersistent(Object)	
boolean isTransactional(Object)	

- obtenir des objets de l'implémentation JDO

Nom	Rôle
PersistenceManager getPersistenceManager(Object)	renvoie l'objet de type PersistenceManager utilisé pour rendre persistante l'instance JDO fournie en paramètre
getObjectId(Object)	
makeDirty(Object, String)	
PersistenceManagerFactory getPersistenceManagerFactory(Properties)	

31.4. La mise en oeuvre

La mise en oeuvre de JDO requiert plusieurs étapes :

- définition d'une classe qui va encapsuler les données (instance JDO)
- définition d'une classe qui va utiliser les données
- compilation des deux classes
- définition d'un fichier de description
- enrichissement de la classe qui va contenir les données

31.4.1. Définition d'une classe qui va encapsuler les données

Une telle classe se présente sous la forme d'une bean : elle représente une occurence particulière dans le système de stockage des données.

Cette classe n'a pas besoin ni d'utiliser ni d'importer de classes de l'API JDO.

Pour la classe qui va contenir des données, JDO impose la présence d'un constructeur sans argument. Celui ci est automatiquement ajouté à la compilation si aucun autre constructeur n'est défini, sinon il faut ajouter un constructeur sans argument manuellement.

31.4.2. Définition d'une classe qui va utiliser les données

Cette classe va réaliser des traitements en utilisant JDO pour accéder et/ou mettre à jour des données.

La suite de ce section sera développée dans une version future de ce document

31.4.3. Compilation des classes

Toutes les classes écrites doivent être compilées normalement comme toutes classes Java.

31.4.4. Définition d'un fichier de description

Pour indiquer à JDO quelles classes doivent être persistantes et préciser des informations concernant ces dernières, il faut utiliser un fichier particulier au format XML. Ce fichier désigné par "Metadata" dans les spécifications doit avoir pour extension .jdo.

Il est possible de définir un fichier de description pour chaque classes persistantes ou un fichier pour un package concernant toutes les classes persistantes du package. Dans le premier cas, le fichier doit se nommer nom_de_la_classe.jdo, dans le second nom_du_package.jdo.

Le fichier commence par un prologue :

<?xml version="1.0" encoding="UTF−8" ?>

Le fichier contient ensuite la DTD utilisée pour valider le fichier : soit une URL pointant sur la DTD du site de Sun soit une DTD sur le système de fichier.

<!DOCTYPE jdo PUBLIC "−//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//EN" "http://java.sun.com/dtd/jdo_1_0.dtd">

Le tag racine du document XML est <jdo>. Ce tag peut contenir un ou plusieurs tags <package> selon les besoins, chaque tag package concernant un seul package.

Chaque tag <package> contient autant de tags <class> que de classes de type instance JDO utilisées. L'attribut "name", obligatoire, permet de préciser le nom de la classe.

Les tags <jdo>, <package>, <class> et <field> peuvent aussi avoir un tag <extension> qui va contenir des paramètres particuliers dédiés à l'implémentation de JDO utilisée. Il faut un tag <extension> pour chaque implémentation utilisée.

31.4.5. Enrichissement de la classe qui va contenir les données

Cette phase permet d'ajouter du code à chaque classe encapsulant une instance JDO. Ce code contient les méthodes définies par l'interface PersistenceCapable.

Le ou les outils fournis par le fournisseur sont particuliers pour chaque implémentation utilisée.

31.5. Parcours de toutes les occurences

Un objet qui implémente l'interface Extent permet d'accéder à toutes les instances d'une classe encapsulant des données.

Un objet de type Extent est obtenu en appellant la méthode getExtent() d'un objet PersistentManager. Cette méthode attend deux paramètres : un objet de type Class qui est la classe encapsulant les données et un booléen qui permet de préciser si les sous classes doivent être prise en compte.

Un objet de type Extent ne permet qu'une seule opération sur l'ensemble des instances qu'il contient : obtenir un objet de type Iterator qui permet le parcours séquentiel de toutes les occurences. La méthode iterator() permet de renvoyer cet objet de type Iterator : les méthodes hasNext() et next() assurent le parcours des occurences.

L'appel de la méthode close() une fois que l'objet de type Iterator fourni en paramètre n'a plus d'utilité est obligatoire pour permettre de liberer les ressources allouées par l'objet pour son fonctionnement. La méthode closeAll() permet de fermer tout les objets de type Iterator instanciés par l'objet de type Extent.

Exemple : Afficher tous les données de la table personne

```
package testjdo;

import javax.jdo.*;
import java.util.*;

public class PersonneExtent {

  private PersistenceManagerFactory pmf = null;
  private PersistenceManager pm = null;

  public PersonneExtent() {
    try {

      pmf =
        (PersistenceManagerFactory) (Class
          .forName("com.libelis.lido.PersistenceManagerFactory")
          .newInstance());
      pmf.setConnectionDriverName("org.gjt.mm.mysql.Driver");
      pmf.setConnectionURL("jdbc:mysql://localhost/testjdo");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void afficherTous() {
    pm = pmf.getPersistenceManager();

    Extent personneExtent = pm.getExtent(Personne.class, true);

    Iterator iter = personneExtent.iterator();
    while (iter.hasNext()) {
      Personne personne = (Personne) iter.next();
      System.out.println(personne.getNom() + " " + personne.getPrenom());
    }
    personneExtent.close(iter);
  }

  public static void main(String args[]) {
    PersonneExtent pe = new PersonneExtent();
    pe.afficherTous();
  }
}
```

Exemple : Contenu de la table au moment de l'exécution

```
C:\mysql\bin>mysql testjdo
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.16-nt
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> select * from t_personne;
+--------+---------+------------+---------------------+
| LIDOID | nom     | prenom     | datenaiss           |
+--------+---------+------------+---------------------+
|      1 | mon nom | mon prenom | 2004-01-23 20:06:11 |
|   1025 | Nom1    | Jean       | 2004-02-10 00:20:39 |
|   2049 | Nom4    | Jean       | 2004-02-10 00:25:09 |
|   3073 | Nom3    | Louis      | 2004-02-10 21:36:32 |
+--------+---------+------------+---------------------+
4 rows in set (0.00 sec)

mysql>
```

Exemple : Contenu de la table au moment de l'exécution

```
Execution du test
mon nom mon prenom
Nom1 Jean
Nom4 Jean
Nom3 Louis
Execution terminee
```

31.6. La mise en oeuvre de requêtes

Avec JDO, les requêtes sont mises en oeuvre grâce à un objet de type Query. Les requêtes appliquent un filtre sur un ensemble d'objets encapsulant des données sous la forme d'un objet de type Extent ou d'une collection.

Un filtre est une expression booléenne appliquée à chacune des occurences : la requête renvoie toutes les occurences pour laquelle le résultat de l'évaluation de l'expression est vrai. Les expressions sont exprimées avec un langage particulier nommé JDO Query Langage (JDOQL)

Un instance d'un objet qui implemente l'interface Query est obtenu en utilisant la méthode newQuery() d'un objet de type PersistanceManager.

Exemple : afficher les occurences dont le prénom est Jean

```
package testjdo;

import javax.jdo.*;
import java.util.*;

public class PersonneQuery {

  private PersistenceManagerFactory pmf = null;
  private PersistenceManager pm = null;

  public PersonneQuery() {
    try {

      pmf =
        (PersistenceManagerFactory) (Class
          .forName("com.libelis.lido.PersistenceManagerFactory")
          .newInstance());
      pmf.setConnectionDriverName("org.gjt.mm.mysql.Driver");
      pmf.setConnectionURL("jdbc:mysql://localhost/testjdo");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void filtrer() {
```

```
    pm = pmf.getPersistenceManager();

    Extent personneExtent = pm.getExtent(Personne.class, true);
    String filtre = "prenom == \"Jean\"";

    Query query = pm.newQuery(personneExtent, filtre);
    query.setOrdering("nom ascending, prenom ascending");
    Collection result = (Collection) query.execute();

    Iterator iter = result.iterator();
    while (iter.hasNext()) {
      Personne personne = (Personne) iter.next();
      System.out.println(personne.getNom() + " " + personne.getPrenom());
    }
    query.close(result);
  }

  public static void main(String args[]) {
    PersonneQuery pq = new PersonneQuery();
    pq.filtrer();
  }
}
```

Résultat :

```
Execution du test
Nom1 Jean
Nom4 Jean
Execution terminee
```

La suite de ce chapitre sera développée dans une version future de ce document

32. D'autres solutions de mapping objet−relationnel

En plus des API standardisées tel que JDO ou les EJB de type CMP, il existe plusieurs solutions de mapping objet−relationnel (Object Relational Mapping) développées par des entités commerciales ou libres afin de pallier à un besoin longtemps resté sans standard.

Un projet open source est présenté dans ce chapitre : Hibernate.

Hibernate a plusieurs avantages :

- manipulation de données d'une base de données relationnelles à partir d'objets Java
- open source
- facile à mettre en oeuvre, efficace et fiable

32.1. Hibernate

Hibernate est un projet open source visant à proposer un outil de mapping entre les objets et des données stockées dans une base de données relationnelle. Ce projet ne repose sur aucun standard mais il est très populaire notamment à cause de ses bonnes performances et de son ouverture avec de nombreuses bases de données.

Les bases de données supportées sont les principales du marché : DB2, Oracle, MySQL, PostgreSQL, Sybase, SQL Server, Sap DB, Interbase, ...

Le site officiel http://www.hibernate.org contient beaucoup d'informations sur l'outil et propose de le télécharger ainsi que sa documentation.

La version utilisée dans cette section est la 2.1.2 : il faut donc télécharger le fichier hibernate−2.1.2.zip et le décompresser dans un répertoire du système.

Cette section va utiliser Hibernate avec une base de données de type MySQL possédant une table nommée "personnes".

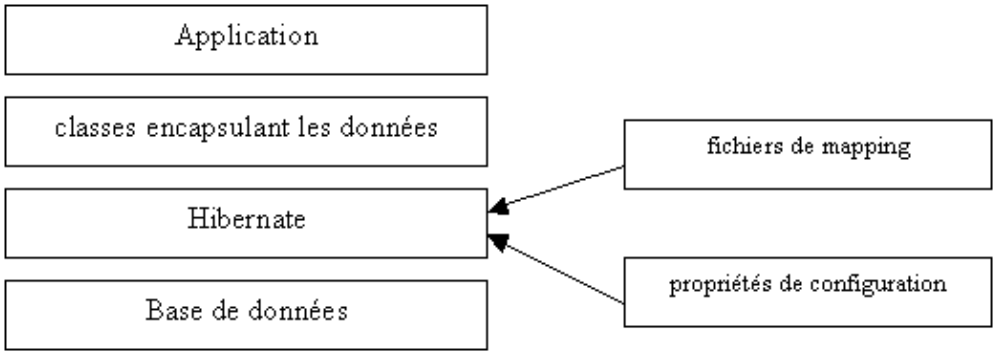
personnes

Field name	Field type
idpersonne *	int(11)
prenompersonne	varchar(50)
nompersonne	varchar(50)
datenaisspersonne	datetime

Hibernate a besoin de plusieurs éléments pour fonctionner :

- une classe de type javabean qui encapsule les données d'une occurence d'une table
- un fichier de correspondance qui configure la correspondance entre la classe et la table
- des propriétés de configuration notamment des informations concernant la connexion à la base de données

Une fois ces éléments correctement définis, il est possible d'utiliser Hibernate dans le code des traitements à réaliser. L'architecture d'Hibernate est donc la suivante :



Cette section survole uniquement les principales fonctionnalités d'Hibernate qui est un outil vraiment complet : pour de plus amples informations, il est nécessaire de consulter la documentation officielle fournie avec l'outil ou consultable sur le site web.

32.1.1. La création d'une classe qui va encapsuler les données

Cette classe doit respecter le standard des javabeans notamment encapsuler les propriétés dans ces champs private avec des getters et setters et avoir un constructeur par défaut.

Les types utilisables pour les propriétés sont : les types primitifs, les classes String et Dates, les wrappers, et n'importe quelle classe qui encapsule une autre table ou une partie de la table.

Exemple :

```
import java.util.Date;

public class Personnes {

  private Integer idPersonne;
  private String nomPersonne;
  private String prenomPersonne;
  private Date datenaissPersonne;

  public Personnes(String nomPersonne, String prenomPersonne, Date datenaissPersonne) {
    this.nomPersonne = nomPersonne;
    this.prenomPersonne = prenomPersonne;
    this.datenaissPersonne = datenaissPersonne;
  }

  public Personnes() {
  }

  public Date getDatenaissPersonne() {
    return datenaissPersonne;
  }

  public Integer getIdPersonne() {
    return idPersonne;
  }

  public String getNomPersonne() {
    return nomPersonne;
  }

  public String getPrenomPersonne() {
    return prenomPersonne;
  }

  public void setDatenaissPersonne(Date date) {
    datenaissPersonne = date;
  }

  public void setIdPersonne(Integer integer) {
    idPersonne = integer;
```

```
  }

  public void setNomPersonne(String string) {
    nomPersonne = string;
  }

  public void setPrenomPersonne(String string) {
    prenomPersonne = string;
  }

}
```

32.1.2. La création d'un fichier de correspondance

Pour assurer le mapping, Hibernate a besoin d'un fichier de correspondance (mapping file) au format XML qui va contenir des informations sur la correspondance entre la classe définie et la table de la base de données.

Même si cela est possible, il n'est pas recommandé de définir un fichier de mapping pour plusieurs classes. Le plus simple est de définir un fichier de mapping par classe, nommé du nom de la classe suivi par ".hbm.xml". Ce fichier doit être situé dans le même répertoire que la classe correspondante ou dans la même archive pour les applications packagées.

Différents éléments sont précisés dans ce document XML :

- la classe qui va encapsuler les données
- l'identifiant dans la base de données et son mode de génération
- le mapping entre les propriétés de classe et les champs de la base de données
- les relations
- ...

Le fichier débute par un prologue et une définition de la DTD utilisée par le fichier XML.

Exemple :

```
<?xml version="1.0"?>
  <!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
```

La tag racine du document XML est le tag <hibernate−mapping>. Ce tag peut contient un ou plusieurs tag <class> : il est cependant préférable de n'utiliser qu'un seul tag <class> et de définir autant de fichiers de correspondance que de classes.

Exemple :

Exemple :

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
  <class name="Personnes" table="personnes">
    <id name="idPersonne" type="int" column="idpersonne">
      <generator class="native"/>
    </id>
    <property name="nomPersonne" type="string" not-null="true" />
    <property name="prenomPersonne" type="string" not-null="true" />
    <property name="datenaissPersonne" type="date">
      <meta attribute="field-description">date de naissance</meta>
    </property>
  </class>
</hibernate-mapping>
```

Le tag <class> permet de préciser des informations sur la classe qui va encapsuler les données.

Ce tag possède plusieurs attributs dont les principaux sont:

Nom	Obligatoire	Rôle
name	oui	nom pleinement qualifié de la classe
table	oui	nom de la table dans la base de données
dynamic−update	non	booléen qui indique de ne mettre à jour que les champs dont la valeur a été modifiée (false par défaut)
dynamic−insert	non	booléen qui indique de générer un ordre insert que pour les champs dont la valeur est non nulle (false par défaut)
mutable	non	booléen qui indique si les occurrences peuvent être mises à jour (true par défaut)

Le tag enfant <id> du tag <class> permet de fournir des informations sur l'identifiant d'une occurrence dans la table.

Ce tag possède plusieurs attributs :

Nom	Obligatoire	Rôle
name	non	nom de la propriété dans la classe
type	non	le type Hibernate
column	non	le nom du champ dans la base de données (par défaut le nom de la propriété)
unsaved−value	non	permet de préciser la valeur de l'identifiant pour une instance non encore enregistrée dans la base de données. Les valeurs possibles sont : any, none, null ou une valeur fournie. Null est la valeur par défaut.

Le tag <generator>, fils obligatoire du tag <id>, permet de préciser quel est le mode de génération d'un nouvel identifiant.

Ce tag possède un attribut :

Attribut	Obligatoire	Rôle
class	oui	précise la classe qui va assurer la génération de la valeur d'un nouvel identifiant. Il existe plusieurs classes fournies en standard par Hibernate qui possèdent un nom utilisable comme valeur de cet attribut.

Les classes de génération fournies en standard par Hibernate possèdent chacun un nom :

Nom	Rôle
increment	incrémentation d'une valeur dans la JVM
identity	utilisation d'un identifiant auto−incrementé pour les bases de données qui le supportent (DB2, MySQL, SQL Server, ...)
sequence	utilisation d'une séquence pour les bases de données qui le supportent (Oracle, DB2, PostgreSQL, ...)
hilo	utilisation d'un algorithme qui utilise une valeur réservée pour une table d'une base de données (par exemple une table qui stocke la valeur du prochain identifiant pour chaque table)
seqhilo	idem mais avec une mécanisme proche d'une séquence
uuid.hex	utilisation d'un algorithme générant un identifiant de type UUID sur 32 caractères prenant en compte entre autre l'adresse IP de la machine et l'heure du système
uuid.string	idem générant un identifiant de type UUID sur 16 caractères

native	utilise la meilleure solution proposée par la base de données
assigned	la valeur est fournie par l'application
foreign	la valeur est fournie par un autre objet avec lequel la classe est associée

Certains modes de génération nécessitent des paramètres : dans ce cas, il faut les définir en utilisant un tag fils <param> pour chaque paramètre.

Le tag <property>, fils du tag <class>, permet de fournir des informations sur une propriété et sa correspondance avec un champ dans la base de données.

Ce tag possède plusieurs attributs dont les principaux sont :

Nom	Obligatoire	Rôle
name	oui	précise le nom de la propriété
type	non	précise le type
column	non	précise le nom du champ dans la base de données (par défaut le nom de la propriété)
update	non	précise si le champ est mis à jour lors d'une opération SQL de type update (par défaut true)
insert	non	précise si le champ est mis à jour lors d'une opération SQL de type insert (par défaut true)

Le type doit être soit un type Hibernate (integer, string, date, timestamp, ...), soit les types primitifs java et certaines classes de base (int, java.lang.String, float, java.util.Date, ...), soit une classe qui encapsule des données à rendre persistante.

Le fichier de correspondance peut aussi contenir une description des relations qui existent avec la table dans la base de données.

32.1.3. Les propriétés de configuration

Pour exécuter Hibernate, il faut lui fournir un certain nombre de propriétés concernant sa configuration pour qu'il puisse se connecter à la base de données.

Ces propriétés peuvent être fournies sous plusieurs formes :

- un fichier de configuration nommé hibernate.properties et stocké dans un répertoire inclus dans le classpath
- un fichier de configuration au format XML nommé hibernate.cfg.xml
- utiliser la méthode setProperties() de la classe Configuration
- définir des propriétés dans la JVM en utilisant l'option −Dpropriété=valeur

Les principales propriétés pour configurer la connexion JDBC sont :

Nom de la propriété	Rôle
hibernate.connection.driver_class	nom pleinement qualifié de classe du pilote JDBC
hibernate.connection.url	URL JDBC désignant la base de données
hibernate.connection.username	nom de l'utilisateur pour la connexion
hibernate.connection.password	mot de passe de l'utilisateur
hibernate.connection.pool_size	nombre maximum de connexions dans le pool

Les principales propriétés pour configurer une source de données (DataSource) à utiliser sont :

Nom de la propriété	Rôle
hibernate.connection.datasource	nom du DataSourc enregistré dans JNDI
hibernate.jndi.url	URL du fournisseur JNDI
hibernate.jndi.class	classe pleinement qualifiée de type InitialContextFactory permettant l'accès à JNDI
hibernate.connection.username	nom de l'utilisateur de la base de données
hibernate.connection.password	mot de passe de l'utilisateur

Les principales autres propriétés sont :

Nom de la propriété	Rôle
hibernate.dialect	nom de la classe pleinement qualifiée qui assure le dialogue avec la base de données
hibernate.jdbc.use_scrollable_resultset	booléen qui permet le parcours dans les deux sens pour les connexions fournies à Hibernate utilisant pilotes JDBC 2 supportant cette fonctionnalité
hibernate.show_sql	booléen qui précise si les requêtes SQL générées par Hibernate sont affichées dans la console (particulièrement utile lors du débogage)

Hibernate propose des classes qui héritent de la classe Dialect pour chaque base de données supportée. C'est le nom de la classe correspondant à la base de données utilisées qui doit être obligatoirement fourni à la propriété hibernate.dialect.

Pour définir les propriétés utiles, le plus simple est de définir un fichier de configuration qui en standard doit se nommer hibernate.properties. Ce fichier contient des paires clé=valeur pour chaque propriété définie.

Exemple : paramètres pour utiliser une base de données MySQL

```
hibernate.dialect=net.sf.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost/testDB
hibernate.connection.username=root
hibernate.connection.password=
```

Le pilote de la base de données utilisée, mysql−connector−java−3.0.11−stable−bin.jar dans l'exemple, doit être ajouté dans le classpath.

Il est aussi possible de définir les propriétés dans un fichier au format XML nommé en standard hibernate.cfg.xml

Les propriétés sont alors définies par un tag <property>. Le nom de la propriété est fourni grace à l'attribut « name » et sa valeur est fourni dans le corps du tag.

Exemple :

```
<?xml version='1.0' encoding='utf−8'?>
<!DOCTYPE hibernate−configuration PUBLIC "−//Hibernate/Hibernate Configuration DTD 2.0//EN"
  "http://hibernate.sourceforge.net/hibernate−configuration−2.0.dtd">
<hibernate−configuration>
  <session−factory>
    <property name="hibernate.dialect">net.sf.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost/testDB</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password"></property>
    <property name="dialect">net.sf.hibernate.dialect.MySQLDialect</property>
    <property name="show_sql">true</property>
    <mapping resource="Personnes.hbm.xml"/>
  </session−factory>
```

```
</hibernate-configuration>
```

32.1.4. L'utilisation d'Hibernate

Pour utiliser Hibernate dans le code, il est nécessaire de réaliser plusieurs opérations :

- création d'une instance de la classe
- création d'une instance de la classe SessionFactory
- création d'une instance de la classe Session qui va permettre d'utiliser les services d'Hibernate

Si les propriétés sont définies dans le fichier hibernate.properties, il faut tout d'abord créer une instance de la classe Configuration. Pour lui associer la ou les classes encapsulant les données, la classe propose deux méthodes :

- addFile() qui attend en paramètre le nom du fichier de mapping
- addClass() qui attend en paramètre un objet de type Class encapsulant la classe. Dans ce cas, la méthode va rechercher un fichier nommé nom_de_la_classe.hbm.xml dans le classpath (ce fichier doit se situe dans le même répertoire que le fichier .class de la classe correspondante)

Une instance de la classe Session est obtenu à partir d'une fabrique de type SessionFactory. Cet objet est obtenu à partir de l'instance du type Configuration en utilisant la méthode buildSessionFactory().

La méthode openSession() de la classe SessionFactory permet d'obtenir une instance de la classe Session.

Par défaut, la méthode openSession() qui va ouvrir une connexion vers la base de données en utilisant les informations fournies par les propriétés de configuration.

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.Date;

public class TestHibernate1 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();
...
```

Il est aussi possible de fournir en paramètre de la méthode openSession() une instance de la classe javax.sql.Connection qui encapsule la connexion à la base de données.

Pour une utilisation du fichier hibernate.cfg.xml, il faut créer une occurrence de la classe Configuration, appeler sa méthode configure() qui va lire le fichier XML et appeler la méthode buildSessionFactory() de l'objet renvoyer par la méthode configure().

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.*;

public class TestHibernate1 {
  public static void main(String args[]) throws Exception {
    SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
...
```

Il est important de clôturer l'objet Session, une fois que celui est devenu inutile, en utilisant la méthode close().

32.1.5. La persistance d'une nouvelle occurence

Pour créer une nouvelle occurrence dans la source de données, il suffit de créer une nouvelle instance de classe encapsulant les données, de valoriser ces propriétés et d'appeler la méthode save() de la session en lui passant en paramètre l'objet encapsulant les données.

La méthode save() n'a aucune action directe sur la base de données. Pour enregistrer les données dans la base, il faut réaliser un commit sur la connexion ou la transaction ou faire appel à la méthode flush() de la classe Session.

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.Date;

public class TestHibernate1 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();

    Transaction tx = null;
    try {
      tx = session.beginTransaction();
      Personnes personne = new Personnes("nom3", "prenom3", new Date());
      session.save(personne);
      session.flush() ;
      tx.commit();
    } catch (Exception e) {
      if (tx != null) {
        tx.rollback();
      }
      throw e;
    } finally {
      session.close();
    }

    sessionFactory.close();
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate1
Buildfile: build.xml
init:
[copy] Copying 1 file to C:\java\test\testhibernate\bin
compile:
TestHibernate1:
[java] 12:41:37,402 INFO Environment:462 - Hibernate 2.1.2
[java] 12:41:37,422 INFO Environment:496 - loaded properties from resource
hibernate.properties: {hibernate.connection.username=root, hibernate.connection
.password=, hibernate.cglib.use_reflection_optimizer=true, hibernate.dialect=net
.sf.hibernate.dialect.MySQLDialect, hibernate.connection.url=jdbc:mysql://localh
ost/testDB, hibernate.connection.driver_class=com.mysql.jdbc.Driver}
[java] 12:41:37,432 INFO Environment:519 - using CGLIB reflection optimize
r
[java] 12:41:37,502 INFO Configuration:329 - Mapping resource: Personnes.h
bm.xml
[java] 12:41:38,784 INFO Binder:229 - Mapping class: Personnes -> personne
s
[java] 12:41:38,984 INFO Configuration:595 - processing one-to-many associ
ation mappings
[java] 12:41:38,994 INFO Configuration:604 - processing one-to-one associa
tion property references
[java] 12:41:38,994 INFO Configuration:629 - processing foreign key constr
aints
```

```
[java] 12:41:39,074 INFO Dialect:82 − Using dialect: net.sf.hibernate.dial
ect.MySQLDialect
[java] 12:41:39,084 INFO SettingsFactory:62 − Use outer join fetching: tru
e
[java] 12:41:39,104 INFO DriverManagerConnectionProvider:41 − Using Hibern
ate built−in connection pool (not for production use!)
[java] 12:41:39,114 INFO DriverManagerConnectionProvider:42 − Hibernate co
nnection pool size: 20
[java] 12:41:39,144 INFO DriverManagerConnectionProvider:71 − using driver
: com.mysql.jdbc.Driver at URL: jdbc:mysql://localhost/testDB
[java] 12:41:39,154 INFO DriverManagerConnectionProvider:72 − connection p
roperties: {user=root, password=}
[java] 12:41:39,185 INFO TransactionManagerLookupFactory:33 − No Transacti
onManagerLookup configured (in JTA environment, use of process level read−write
cache is not recommended)
[java] 12:41:39,625 INFO SettingsFactory:102 − Use scrollable result sets:
true
[java] 12:41:39,635 INFO SettingsFactory:105 − Use JDBC3 getGeneratedKeys(
): true
[java] 12:41:39,635 INFO SettingsFactory:108 − Optimize cache for minimal
puts: false
[java] 12:41:39,635 INFO SettingsFactory:117 − Query language substitution
s: {}
[java] 12:41:39,645 INFO SettingsFactory:128 − cache provider: net.sf.ehca
che.hibernate.Provider
[java] 12:41:39,685 INFO Configuration:1080 − instantiating and configurin
g caches
[java] 12:41:39,946 INFO SessionFactoryImpl:119 − building session factory
[java] 12:41:41,237 INFO SessionFactoryObjectFactory:82 − no JNDI name con
figured
[java] 12:41:41,768 INFO SessionFactoryImpl:531 − closing
[java] 12:41:41,768 INFO DriverManagerConnectionProvider:137 − cleaning up
connection pool: jdbc:mysql://localhost/testDB
BUILD SUCCESSFUL
Total time: 7 seconds
C:\java\test\testhibernate>
```

32.1.6. Obtenir une occurrence à partir de son identifiant

La méthode load() de la classe Session permet d'obtenir une instance de la classe des données encapsulant les données de l'occurrence de la base dont l'identifiant est fourni en paramètre.

Il existe deux surcharges de la méthode :

- la première attend en premier paramètre le type de la classe des données et renvoie une nouvelle instance de cette classe
- la seconde attend en paramètre une instance de la classe des données et la met à jour avec les données retrouvées

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;

public class TestHibernate2 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();

    try {
      Personnes personne = (Personnes) session.load(Personnes.class, new Integer(3));
      System.out.println("nom = " + personne.getNomPersonne());
    } finally {
      session.close();
    }

    sessionFactory.close();
```

```
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate2
Buildfile: build.xml

init:

compile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin

TestHibernate2:
     [java] nom = nom3

BUILD SUCCESSFUL
Total time: 9 seconds
```

32.1.7. Le langage de requête HQL

Pour offrir un langage d'interrogation commun à toute les base de données, Hibernate propose son propre langage nommé HQL (Hibernate Query Language)

Le langage HQL est proche de SQL avec une utilisation sous forme d'objets des noms de certaines entités : il n'y a aucune référence aux tables ou aux champs car ceux ci sont référencés respectivement par leur classe et leurs propriétés. C'est Hibernate qui se charge de générer la requête SQL à partir de la requête HQL en tenant compte du contexte (type de base de données utilisée défini dans le fichier de configuration et la configuration du mapping).

La méthode find() de la classe Session permet d'effectuer une recherche d'occurrences grâce à la requête fournie en paramètre.

Exemple : rechercher toutes les occurrences d'une table

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.*;

public class TestHibernate3 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();

    try {
      List personnes = session.find("from Personnes");
      for (int i = 0; i < personnes.size(); i++) {
        Personnes personne = (Personnes) personnes.get(i);
        System.out.println("nom = " + personne.getNomPersonne());
      }
    } finally {
      session.close();
    }
    sessionFactory.close();
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate3
Buildfile: build.xml

init:
```

```
compile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin

TestHibernate3:
     [java] nom = nom1
     [java] nom = nom2
     [java] nom = nom3

BUILD SUCCESSFUL
Total time: 14 seconds
```

La méthode find() possède deux surcharges pour permettre de fournir un seul ou plusieurs paramètres dans la requête.

La première surcharge permet de fournir un seul paramètre : elle attend en paramètre la requête, la valeur du paramètre et le type du paramètre.

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.*;

public class TestHibernate4 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();

    try {
      List personnes = session.find("from Personnes p where p.nomPersonne=?",
        "nom1", Hibernate.STRING);
      for (int i = 0; i < personnes.size(); i++) {
        Personnes personne = (Personnes) personnes.get(i);
        System.out.println("nom = " + personne.getNomPersonne());
      }
    } finally {
      session.close();
    }

    sessionFactory.close();
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate4
Buildfile: build.xml

init:

compile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin

TestHibernate4:
     [java] nom = nom1

BUILD SUCCESSFUL
```

Dans la requête du précédent exemple, un alias nommé « p » est défini pour la classe Personnes. Le mode de fonctionnement d'un alias est similaire en HQL et en SQL.

La classe Session propose une méthode iterate() dont le mode de fonctionnement est similaire à la méthode find() mais elle renvoie un iterateur (objet de type Iterator) sur la collection des éléments retrouvés plutôt que la collection elle

même.

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.*;

public class TestHibernate6 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();

    try {
      Iterator personnes = session.iterate("from Personnes ");
      while (personnes.hasNext()) {
        Personnes personne = (Personnes) personnes.next();
        System.out.println("nom = " + personne.getNomPersonne());
      }
    } finally {
      session.close();
    }

    sessionFactory.close();
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate6
Buildfile: build.xml

init:

compile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin

TestHibernate6:
     [java] nom = nom1
     [java] nom = nom2
     [java] nom = nom3

BUILD SUCCESSFUL
```

Il est aussi possible d'utiliser la clause « order by » dans une requête HQL pour définir l'ordre de tri des occurrences.

Exemple :

List personnes = session.find("from Personnes p order by p.nomPersonne desc");

Il est possible d'utiliser des fonctions telles que count() pour compter le nombre d'occurrences.

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.*;

public class TestHibernate5 {

  public static void main(String args[]) throws Exception {
    Configuration config = new Configuration();
    config.addClass(Personnes.class);
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();
```

```
    try {
      int compteur = ( (Integer) session.iterate(
        "select count(*) from Personnes").next() ).intValue();
      System.out.println("compteur = " + compteur);
    } finally {
      session.close();
    }

    sessionFactory.close();
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate5
Buildfile: build.xml

init:

compile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin

TestHibernate5:
     [java] compteur = 3

BUILD SUCCESSFUL
```

Il est également possible de définir des requêtes utilisant des paramètres nommés grâce à un objet implémentant l'interface Query. Un objet de type Query est obtenu en invoquant la méthode createQuery() de la classe Session avec comme paramètre la requête HQL.

Dans cette requête, les paramètres sont précisés avec un caractère « : » suivi d'un nom unique.

L'interface Query propose de nombreuses méthodes setXXX() pour associer à chaque paramètre une valeur en fonction du type de la valeur (XXX représente le type). Chacune de ces méthodes possède deux surcharges permettant de préciser le paramètre (à partir de son nom ou de son index dans la requête) et sa valeur.

Pour parcourir la collection des occurrences trouvées, l'interface Query propose la méthode list() qui renvoie une collection de type List ou la méthode iterate() qui renvoie un itérateur sur la collection.

Exemple :

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
import java.util.*;

public class TestHibernate8 {

  public static void main(String args[]) throws Exception {
    SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
    Session session = sessionFactory.openSession();

    try {
      Query query = session.createQuery("from Personnes p where p.nomPersonne = :nom");
      query.setString("nom", "nom2");
      Iterator personnes = query.iterate();

      while (personnes.hasNext()) {
        Personnes personne = (Personnes) personnes.next();
        System.out.println("nom = " + personne.getNomPersonne());
      }
    } finally {
      session.close();
    }

    sessionFactory.close();
  }
}
```

Résultat :

```
C:\java\test\testhibernate>ant TestHibernate8
Buildfile: build.xml

init:
     [copy] Copying 1 file to C:\java\test\testhibernate\bin

compile:

TestHibernate8:
     [java] nom = nom2

BUILD SUCCESSFUL
Total time: 7 seconds
```

Hibernate propose également d'externaliser une requête dans le fichier de mapping.

32.1.8. La mise à jour d'une occurence

Pour mettre à jour une occurrence dans la source de données, il suffit d'appeler la méthode update() de la session en lui passant en paramètre l'objet encapsulant les données.

Le mode de fonctionnement de cette méthode est similaire à celui de la méthode save().

La méthode saveOrUpdate() laisse Hibernate choisir entre l'utilisation de la méthode save() ou update() en fonction de la valeur de l'identifiant dans la classe encapsulant les données.

32.1.9. La suppression d'une ou plusieurs occurrences

La méthode delete() de la classe Session permet de supprimer une ou plusieurs occurrences en fonction de la version surchargée de la méthode utilisée.

Pour supprimer une occurrence encapsulée dans une classe, il suffit d'invoquer la classe en lui passant en paramètre l'instance de la classe.

Pour supprimer plusieurs occurrences, voire toutes, il faut passer en paramètre de la méthode delete(), une chaîne de caractères contenant la requête HQL pour préciser les éléments concernés par la suppression.

Exemple : suppression de toutes les occurrences de la table

session.delete("from Personnes");

32.1.10. Les relations

Un des fondements du modèle de données relationnelles repose sur les relations qui peuvent intervenir entre une table et une ou plusieurs autres tables ou la table elle même.

Hibernate propose de transcrire ces relations du modèle relationnel dans le modèle objet. Il supporte plusieurs types de relations :

- relation de type 1 − 1 (one−to−one)
- relation de type 1 − n (many−to−one)
- relation de type n − n (many−to−many)

Dans le fichier de mapping, il est nécessaire de définir les relations entre la table concernée et les tables avec lesquelles elle possède des relations.

La suite de cette section sera développée dans une version future de ce document

32.1.11. Les outils de génération de code

Hibernate fournit séparemment un certain nombre d'outils. Ces outils sont livrés séparément dans un fichier nommé hibernate−extensions−2.1.zip.

Il faut télécharger et décompresser le contenu de cette archive par exemple dans le répertoire ou Hibernate a été décompressé.

L'archive contient deux répertoires :

- hibern8ide
- tools

Le répertoire tools propose trois outils :

- class2hbm :
- ddl2hbm :
- hbm2java :

Pour utiliser ces outils, il y a deux solutions possibles :

- utiliser les fichiers de commande .bat fourni dans le répertoire /tools/bin
- utiliser ant pour lancer ces outils

Pour utiliser les fichiers de commandes .bat, il est nécessaire au préalable de configurer les paramètres dans le fichier setenv.bat. Il faut notamment correctement renseigner les valeurs associées aux variables JDBC_DRIVER qui précise le pilote de la base de données et HIBERNATE_HOME qui précise le répertoire ou est installé Hibernate.

Pour utiliser les outils dans un script ant, il faut créer ou modifier un fichier build en ajoutant une tâche pour l'outil à utiliser.

Il faut copier le fichier hibernate2.jar et les fichiers contenus dans le répertoire /lib d'Hibernate dans le répertoire lib du projet. Il faut aussi copier dans ce répertoire les fichiers contenus dans le répertoire /tools/lib et le fichier /tools/hibernate−tools.jar.

Pour éviter les messages d'avertissement sur la configuration manquante de log4j, le plus simple est de copier le fichier /src/log4j.properties d'Hibernate dans le répertoire bin du projet.

La suite de ce chapitre sera développée dans une version future de ce document

33. Java et XML

L'utilisation ensemble de Java et XML est facilitée par le fait qu'ils ont plusieurs points communs :

- indépendance de toute plateforme
- conçu pour être utilisé sur un réseau
- prise en charge de la norme Unicode

Ce chapitre contient plusieurs sections :

33.1. Présentation de XML

XML est l'acronyme de «eXtensible Markup Language».

XML permet d'échanger des données entres applications hétérogènes car il permet de modéliser et de stocker des données de façon portable.

XML est extensible dans la mesure ou il n'utilise pas de tags prédéfinis comme HTML et il permet de définir de nouvelles balises : c'est un métalangage.

Le format HTML est utilisé pour formater et afficher les données qu'il contient : il est destiné à structurer, formater et échanger des documents d'une façon la plus standard possible..

XML est utilisé pour modéliser et stocker des données. Il ne permet pas à lui seul d'afficher les données qu'il contient.

Pourtant, XML et HTML sont tous les deux des dérivés d'une langage nommé SGML (Standard Generalized Markup Language). La création d'XML est liée à la complexité de SGML. D'ailleurs, un fichier XML avec sa DTD correspondante peut être traité par un processeur SGML.

XML et java ont en commun la portabilité réalisée grâce à une indépendance vis à vis du système et de leur environnement.

33.2. Les règles pour formater un document XML

Un certain nombre de règles doivent être respectées pour définir un document XML valide et «bien formé». Pour pouvoir être analysé, un document XML doit avoir une syntaxe correcte. Les principales règles sont :

- le document doit contenir au moins une balise
- chaque balise d'ouverture (exemple <tag>) doit posséder une balise de fermeture (exemple </tag>). Si le tag est vide, c'est à dire qu'il ne possède aucune données (exemple <tag></tag>), un tag abrégé peut être utilisé (exemple correspondant : <tag/>)
- les balises ne peuvent pas être intercalées (exemple <liste><element></liste></element> n'est pas autorisé)
- toutes les balises du document doivent obligatoirement être contenues entre une balise d'ouverture et de fermeture unique dans le document nommée élément racine
- les valeurs des attributs doivent obligatoirement être encadrées avec des quotes simples ou doubles
- les balises sont sensibles à la casse
- Les balises peuvent contenir des attributs même les balises vides
- les données incluses entre les balises ne doivent pas contenir de caractères < et & : il faut utiliser respectivement &lt ; et &amp ;
- La première ligne du document devrait normalement correspondre à la déclaration de document XML : le prologue.

33.3. La DTD (Document Type Definition)

Les balises d'un document XML sont libres. Pour pouvoir valider si le document est correct, il faut définir un document nommé DTD qui est optionnel. Sans sa présence, le document ne peut être validé : on peut simplement vérifier que la syntaxe du document est correcte.

Une DTD est un document qui contient la grammaire définissant le document XML. Elle précise notamment les balises autorisées et comment elles s'imbriquent.

La DTD peut être incluse dans l'en tête du document XML ou être mise dans un fichier indépendant. Dans ce cas, la directive < !DOCTYPE> dans le document XML permet de préciser le fichier qui contient la DTD.

Il est possible d'utiliser une DTD publique ou de définir sa propre DTD si aucune ne correspond à ces besoins.

Pour être valide, un document XML doit avoir une syntaxe correcte et correspondre à la DTD.

33.4. Les parseurs

Il existe plusieurs types de parseur. Les deux plus répandus sont ceux qui utilisent un arbre pour représenter et exploiter le document et ceux qui utilisent des événements. Le parseur peut en plus permettre de valider le document XML.

Ceux qui utilisent un arbre permettent de le parcourir pour obtenir les données et modifier le document.
Ceux qui utilisent des événements associent à des événements particuliers des méthodes pour traiter le document.

SAX (Simple API for XML) est une API libre créée par David Megginson qui utilisent les événements pour analyser et exploiter les documents au format XML.

Les parseurs qui produisent des objets composant une arborescence pour représenter le document XML utilisent le modèle DOM (Document Object Model) défini par les recommandations du W3C.

Le choix d'utiliser SAX ou DOM doit tenir compte de leurs points forts et de leurs faiblesses :

	les avantages	les inconvénients
DOM	parcours libre de l'arbre possibilité de modifier la structure et le contenu de l'arbre	gourmand en mémoire doit traiter tout le document avant d'exploiter les résultats
SAX	peut gourmand en ressources mémoire rapide principes faciles à mettre en oeuvre permet de ne traiter que les données utiles	traite les données séquentiellement un peu plus difficile à programmer, il est souvent nécessaire de sauvegarder des informations pour les traiter

SAX et DOM ne fournissent que des définitions : ils ne fournissent pas d'implémentation utilisable. L'implémentation est laissée aux différents éditeurs qui fournissent un parseur compatible avec SAX et/ou DOM.L'avantage d'utiliser l'un deux est que le code utilisé sera compatible avec les autres : le code nécessaire à l'instanciation du parseur est cependant spécifique à chaque fournisseur.

IBM fourni gratuitement un parseur XML : xml4j. Il est téléchargeable à l'adresse suivante : http://www.alphaworks.ibm.com/tech/xml4j

Le groupe Apache développe Xerces à partir de xml4j : il est possible de télécharger la dernière version à l'URL http://xml.apache.org

Sun a développé un projet dénommé Project X. Ce projet a été repris par le groupe Apache sous le nom de Crimson.

Ces trois projets apportent pour la plupart les mêmes fonctionnalitées : ils se distinguent sur des points mineurs : performance, rapidité, facilité d'utilisation etc. ... Ces fonctionnalités évoluent très vite avec les versions de ces parseurs qui se succèdent très rapidement.

Pour les utiliser, il suffit de dézipper le fichier et d'ajouter les fichiers .jar dans la variable définissant le CLASSPATH.

Il existe plusieurs autres parseurs que l'on peut télécharger sur le web.

33.5. L'utilisation de SAX

SAX est l'acronyme de Simple API for XML. Cette API a été développée par David Megginson.

Ce type de parseur utilise des événements pour piloter le traitement d'un fichier XML. Un objet (nommé handler en anglais) doit implémenter des méthodes particulières définies dans une interface de l'API pour fournir les traitements à réaliser : selon les événements, le parseur appelle ces méthodes.

Les dernières informations concernant cette API sont disponible à l'URL : www.megginson.com/SAX/index.html

Les classes de l'API SAX sont regroupées dans le package org.xml.sax

33.5.1. L'utilisation de SAX de type 1

SAX type 1 est composé de deux packages :

- org.xml.sax :
- org.xml.sax.helpers :

SAX définit plusieurs classes et interfaces :

- les interfaces implémentées par le parseur : Parser, AttributeList et Locator
- les interfaces implémentées par le handler : DocumentHandler, ErrorHandler, DTDHandler et EntityHandler
- les classes de SAX :
- des utilitaires rassemblés dans le package org.xml.sax.helpers notamment la classe ParserFactory

Les exemples de cette section utilisent la version 2.0.15 du parseur xml4j d'IBM.

Pour parser un document XML avec un parseur XML SAX de type 1, il faut suivrent les étapes suivantes :

- créer une classe qui implémente l'interface DocumentHandler ou hérite de la classe org.xml.sax.HandlerBase et qui se charge de répondre aux différents événements émis par le parseur
- créer une instance du parseur en utilisant la méthode makeParser() de la classe ParserFactory.
- associer le handler au parseur grâce à la méthode setDocumentHandler()
- exécuter la méthode parse() du parseur

Exemple : avec XML4J

```
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import com.ibm.xml.parsers.*;
import java.io.*;

public class MessageXML {
  static final String DONNEES_XML =
  "<?xml version=\"1.0\"?>\n"
  +"<BIBLIOTHEQUE\n>"
  +" <LIVRE>\n"
  +" <TITRE>titre livre 1</TITRE>\n"
  +" <AUTEUR>auteur 1</AUTEUR>\n"
  +" <EDITEUR>editeur 1</EDITEUR>\n"
  +" </LIVRE>\n"
  +" <LIVRE>\n"
  +" <TITRE>titre livre 2</TITRE>\n"
  +" <AUTEUR>auteur 2</AUTEUR>\n"
  +" <EDITEUR>editeur 2</EDITEUR>\n"
  +" </LIVRE>\n"
  +" <LIVRE>\n"
  +" <TITRE>titre livre 3</TITRE>\n"
  +" <AUTEUR>auteur 3</AUTEUR>\n"
  +" <EDITEUR>editeur 3</EDITEUR>\n"
  +" </LIVRE>\n"
  +"</BIBLIOTHEQUE>\n";

  static final String CLASSE_PARSER = "com.ibm.xml.parsers.SAXParser";

  /**
   * Lance l'application.
   * @param args un tableau d'arguments de ligne de commande
   */
  public static void main(java.lang.String[] args) {

    MessageXML m = new MessageXML();
    m.parse();

    System.exit(0);
  }

  public MessageXML() {
    super();
  }

  public void parse() {
    TestXMLHandler handler = new TestXMLHandler();

    System.out.println("Lancement du parseur");

    try {
      Parser parser = ParserFactory.makeParser(CLASSE_PARSER);

      parser.setDocumentHandler(handler);
      parser.setErrorHandler((ErrorHandler) handler);
```

```
      parser.parse(new InputSource(new StringReader(DONNEES_XML)));

    } catch (Exception e) {
      System.out.println("Exception capturée : ");
      e.printStackTrace(System.out);
      return;
    }
  }
}
```

Il faut ensuite créer la classe du handler.

Exemple :

```
import java.util.*;

/**
 * Classe utilisee pour gérer les evenement emis par SAX lors du traitement du fichier XML
 */
public class TestXMLHandler extends org.xml.sax.HandlerBase {
  public TestXMLHandler() {
    super();
  }

  /**
   * Actions à réaliser sur les données
   */
  public void characters(char[] caracteres, int debut, int longueur) {
    String donnees = new String(caracteres, debut, longueur);
    System.out.println("   valeur = *" + donnees + "*");
  }

  /**
   * Actions à réaliser lors de la fin du document XML.
   */
  public void endDocument() {
    System.out.println("Fin du document");
  }

  /**
   * Actions à réaliser lors de la détection de la fin d'un element.
   */
  public void endElement(String name) {
    System.out.println("Fin tag " + name);
  }

  /**
   * Actions à réaliser au début du document.
   */
  public void startDocument() {
    System.out.println("Debut du document");
  }

  /**
   * Actions a réaliser lors de la detection d'un nouvel element.
   */
  public void startElement(String name, org.xml.sax.AttributeList atts) {
    System.out.println("debut tag : " + name);
  }
}
```

Resultats :

```
Lancement du parser
Debut du document
debut tag : BIBLIOTHEQUE
   valeur = *
  *
```

```
debut tag : LIVRE
   valeur = *
    *
debut tag : TITRE
   valeur = *titre livre 1*
Fin tag TITRE
   valeur = *
    *
debut tag : AUTEUR
   valeur = *auteur 1*
Fin tag AUTEUR
   valeur = *
    *
debut tag : EDITEUR
   valeur = *editeur 1*
Fin tag EDITEUR
   valeur = *
  *
Fin tag LIVRE
   valeur = *
  *
debut tag : LIVRE
   valeur = *
    *
debut tag : TITRE
   valeur = *titre livre 2*
Fin tag TITRE
   valeur = *
    *
debut tag : AUTEUR
   valeur = *auteur 2*
Fin tag AUTEUR
   valeur = *
    *
debut tag : EDITEUR
   valeur = *editeur 2*
Fin tag EDITEUR
   valeur = *
  *
Fin tag LIVRE
   valeur = *
  *
debut tag : LIVRE
   valeur = *
    *
debut tag : TITRE
   valeur = *titre livre 3*
Fin tag TITRE
   valeur = *
    *
debut tag : AUTEUR
   valeur = *auteur 3*
Fin tag AUTEUR
   valeur = *
    *
debut tag : EDITEUR
   valeur = *editeur 3*
Fin tag EDITEUR
   valeur = *
  *
Fin tag LIVRE
   valeur = *
*
Fin tag BIBLIOTHEQUE
Fin du document
```

Un parseur SAX peut créer plusieurs types d'événements dont les principales méthodes pour y répondre sont :

Evénement	Rôle

startElement()	cette méthode est appelée lors de la détection d'un tag de début
endElement()	cette méthode est appelée lors de la détection d'un tag de fin
characters()	cette méthode est appelée lors de la détection de données entre deux tags
startDocument()	cette méthode est appelée lors du début du traitement du document XML
endDocument()	cette méthode est appelée lors de la fin du traitement du document XML

La classe handler doit redéfinir certaines de ces méthodes selon les besoins des traitements.

En règle générale :

- il faut sauvegarder dans une variable le tag courant dans la méthode startElement()
- traiter les données en fonction du tag courant dans la méthode characters()

La sauvegarde du tag courant est obligatoire car la méthode characters() ne contient pas dans ces paramètres le nom du tag correspondant aux données.

Si les données contenues dans le document XML contiennent plusieurs occurrences qu'il faut gérer avec une collection qui contiendra des objets encapsulant les données, il faut :

- gérer la création d'un objet dans la méthode startElement() lors de la rencontre du tag de début d'un nouvel élément de la liste
- alimenter les attributs de l'objet avec les données de chaque tags utiles dans la méthode characters()
- gérer l'ajout de l'objet à la collection dans la méthode endElement() lors de la rencontre du tag de fin d'élément de la liste

La méthode characters() est appelée lors de la détection de données entre un tag de début et un tag de fin mais aussi entre un tag de fin et le tag début suivant lorsqu'il y a des caractères entre les deux. Ces caractères ne sont pas des données mais des espaces, des tabulations, des retour chariots et certains caractères non visibles.

Pour éviter de traiter les données de ces événements, il y a plusieurs solutions :

- supprimer tous les caractères entre les tags : tous les tags et les données sont rassemblés sur une seule et unique ligne. L'inconvénient de cette méthode est que le message est difficilement lisible par un être humain.
- une autre méthode consiste à remettre à vide la donnée qui contient le tag courant (alimentée dans la méthode startElement()) dans la méthode endElement(). Il suffit alors d'effectuer les traitements dans la méthode characters() uniquement si le tag courant est différent de vide

Exemple :

```
import java.util.*;

/**
 * Classe utilisee pour gérer les evenement emis par SAX lors du traitement du fichier XML
 */
public class TestXMLHandler extends org.xml.sax.HandlerBase {
  private String tagCourant = "";
  public TestXMLHandler() {
    super();
  }

  /**
   * Actions à réaliser sur les données
   */
  public void characters(char[] caracteres, int debut, int longueur) {
    String donnees = new String(caracteres, debut, longueur);
    if (!tagCourant.equals("") {
      System.out.println("  Element " + tagCourant
        + ", valeur = *" + donnees + "*");
    }
```

```
  }

  /**
   * Actions à réaliser lors de la fin du document XML.
   */
  public void endDocument() {
    System.out.println("Fin du document");
  }

  /**
   * Actions à réaliser lors de la détection de la fin d'un element.
   */
  public void endElement(String name) {
    tagCourant = "";
    System.out.println("Fin tag " + name);
  }

  /**
   * Actions à réaliser au début du document.
   */
  public void startDocument() {
    System.out.println("Debut du document");
  }

  /**
   * Actions a réaliser lors de la detection d'un nouvel element.
   */
  public void startElement(String name, org.xml.sax.AttributeList atts) {
    tagCourant = name;
    System.out.println("debut tag : " + name);
  }
}
```

Résultat :

```
Lancement du parser
Debut du document
debut tag : BIBLIOTHEQUE
   Element BIBLIOTHEQUE, valeur = *
  *
debut tag : LIVRE
   Element LIVRE, valeur = *
    *
debut tag : TITRE
   Element TITRE, valeur = *titre livre 1*
Fin tag TITRE
debut tag : AUTEUR
   Element AUTEUR, valeur = *auteur 1*
Fin tag AUTEUR
debut tag : EDITEUR
   Element EDITEUR, valeur = *editeur 1*
Fin tag EDITEUR
Fin tag LIVRE
debut tag : LIVRE
   Element LIVRE, valeur = *
    *
debut tag : TITRE
   Element TITRE, valeur = *titre livre 2*
Fin tag TITRE
debut tag : AUTEUR
   Element AUTEUR, valeur = *auteur 2*
Fin tag AUTEUR
debut tag : EDITEUR
   Element EDITEUR, valeur = *editeur 2*
Fin tag EDITEUR
Fin tag LIVRE
debut tag : LIVRE
   Element LIVRE, valeur = *
    *
debut tag : TITRE
   Element TITRE, valeur = *titre livre 3*
Fin tag TITRE
debut tag : AUTEUR
```

```
   Element AUTEUR, valeur = *auteur 3*
Fin tag AUTEUR
debut tag : EDITEUR
   Element EDITEUR, valeur = *editeur 3*
Fin tag EDITEUR
Fin tag LIVRE
Fin tag BIBLIOTHEQUE
Fin du document
```

- enfin il est possible de vérifier si le premier caractère des données contenues en paramètre de la méthode characters() est un caractère de contrôle ou non grâce à la méthode statique isISOControl() de la classe Character

Exemple :

```
...
/**
 * Actions à réaliser sur les données
 */
public void characters(char[] caracteres, int debut, int longueur) {
        String donnees = new String(caracteres, debut, longueur);

        if (!tagCourant.equals("")) {
                if(!Character.isISOControl(caracteres[debut])) {
                        System.out.println("   Element " + tagCourant
                 +", valeur = *" + donnees + "*");
                }
        }
}
...
```

Résultat :

```
Lancement du parser
Debut du document
debut tag : BIBLIOTHEQUE
debut tag : LIVRE
debut tag : TITRE
   Element TITRE, valeur = *titre livre 1*
Fin tag TITRE
debut tag : AUTEUR
   Element AUTEUR, valeur = *auteur 1*
Fin tag AUTEUR
debut tag : EDITEUR
   Element EDITEUR, valeur = *editeur 1*
Fin tag EDITEUR
Fin tag LIVRE
debut tag : LIVRE
debut tag : TITRE
   Element TITRE, valeur = *titre livre 2*
Fin tag TITRE
debut tag : AUTEUR
   Element AUTEUR, valeur = *auteur 2*
Fin tag AUTEUR
debut tag : EDITEUR
   Element EDITEUR, valeur = *editeur 2*
Fin tag EDITEUR
Fin tag LIVRE
debut tag : LIVRE
debut tag : TITRE
   Element TITRE, valeur = *titre livre 3*
Fin tag TITRE
debut tag : AUTEUR
   Element AUTEUR, valeur = *auteur 3*
Fin tag AUTEUR
debut tag : EDITEUR
```

```
   Element EDITEUR, valeur = *editeur 3*
Fin tag EDITEUR
Fin tag LIVRE
Fin tag BIBLIOTHEQUE
Fin du document
```

SAX définit une exception de type SAXParserException lorsque le parseur contient une erreur dans le document en cours de traitement. Les méthodes getLineNumber() et getColumnNumber() permettent d'obtenir la ligne et la colonne ou l'erreur a été détectée.

Exemple :

```
try {
...
}  catch (SAXParseException e) {
  System.out.println("Erreur lors du traitement du document XML");
  System.out.println(e.getMessage());
  System.out.println("ligne : "+e.getLineNumber());
  System.out.println("colonne : "+e.getColumnNumber());
  }
```

Pour les autres erreurs, SAX définit l'exception SAXException.

33.5.2. L'utilisation de SAX de type 2

SAX de type 2 apporte principalement le support des espaces de noms. Les classes et les interfaces sont toujours définies dans les packages org.xml.sax et ses sous packages.

SAX de type 2 définit quatre interfaces que l'objet handler doit ou peut implémenter :

- ContentHandler : interface qui définit les méthodes appellées lors du traitement du document
- ErrorHandler : interface qui définit les méthodes appellées lors du traitement des warnings et de erreurs
- DTDHandler : interface qui définit les méthodes appellées lors du traitement de la DTD
- EntityResolver

Plusieurs classes et interfaces de SAX de type 1 sont deprecated :

	ancienne entité SAX 1	nouvelle entité SAX 2
Interface	org.xml.sax.Parser	XMLReader
	org.xml.sax.DocumentHandler	ContentHandler
	org.xml.sax.AttributeList	Attributes
Classes	org.xml.sax.helpers.ParserFactory	
	org.xml.sax.HandlerBase	DefaultHandler
	org.xml.sax.helpers.AttributeListImpl	AttributesImpl

Les principes de fonctionnement de SAX 2 sont très proche de SAX 1.

Exemple :

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class TestSAX2
{
```

```
  public static void main(String[] args)
  {
    try
    {
      Class c = Class.forName("org.apache.xerces.parsers.SAXParser");
      XMLReader reader = (XMLReader)c.newInstance();
      TestSAX2Handler handler = new TestSAX2Handler();
      reader.setContentHandler(handler);
      reader.parse("test.xml");
    }
    catch(Exception e){System.out.println(e);}
  }

}


class TestSAX2Handler extends DefaultHandler
{
  private String tagCourant = "";

  /**
   * Actions a réaliser lors de la detection d'un nouvel element.
   */
  public void startElement(String nameSpace, String localName,
    String qName, Attributes attr) throws SAXException  {
    tagCourant = localName;
    System.out.println("debut tag : " + localName);
  }

  /**
   * Actions à réaliser lors de la détection de la fin d'un element.
   */
  public void endElement(String nameSpace, String localName,
    String qName) throws SAXException {
    tagCourant = "";
    System.out.println("Fin tag " + localName);
  }

  /**
   * Actions à réaliser au début du document.
   */
  public void startDocument() {
    System.out.println("Debut du document");
  }

  /**
   * Actions à réaliser lors de la fin du document XML.
   */
  public void endDocument() {
    System.out.println("Fin du document");
  }

  /**
   * Actions à réaliser sur les données
   */
  public void characters(char[] caracteres, int debut,
    int longueur) throws SAXException {
    String donnees = new String(caracteres, debut, longueur);

    if (!tagCourant.equals("")) {
      if(!Character.isISOControl(caracteres[debut])) {
        System.out.println("   Element " + tagCourant +",
          valeur = *" + donnees + "*");
      }
    }
  }
}
```

33.6. DOM

DOM est l'acronyme de Document Object Model. C'est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML.

Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours, recherche et mise à jour)

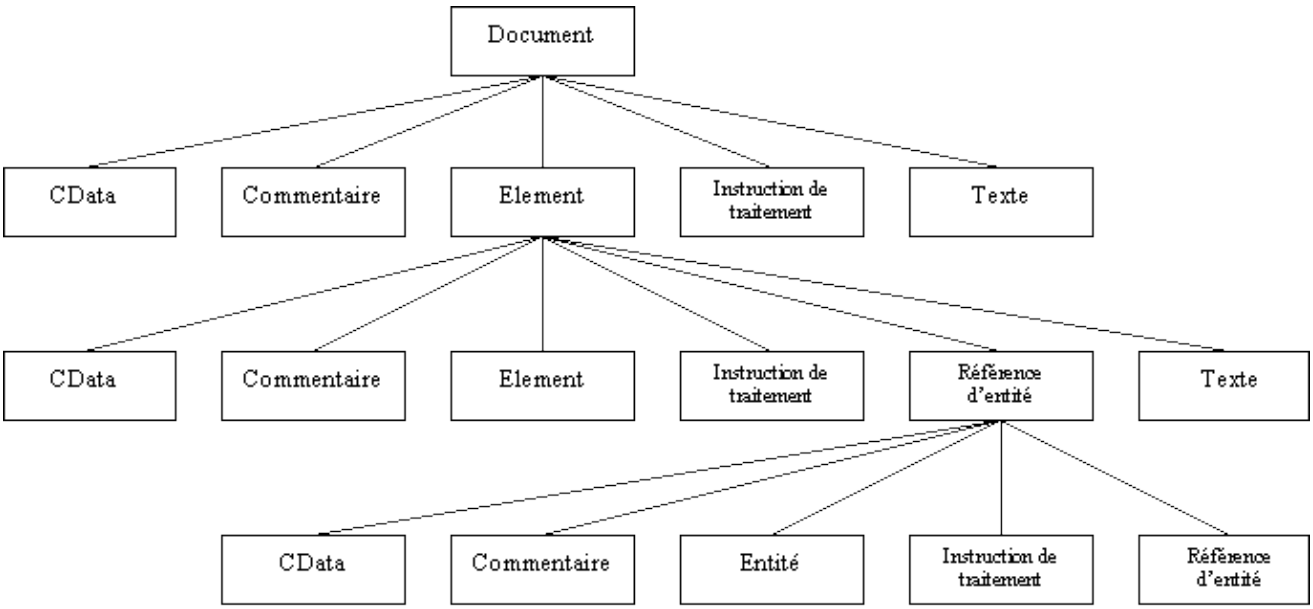
A partir de cette représentation (le modèle), DOM propose de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l'un des aspects les plus intéressant de DOM.

DOM est défini pour être independant du langage dans lequel il sera implémenté. DOM n'est qu'une spécification qui pour être utilisée doit être implémentée par un éditeur tiers.

Il existe deux versions de DOM nommées «niveau» :

- DOM Core Level 1 : cette spécification contient les bases pour manipuler un document XML (document, élément et noeud)
- DOM level 2

Le level 3 est en cours de développement.



Chaque élément qui compose l'arbre possède un type. Selon ce type, l'élément peut avoir certains éléments fils comme le montre le schéma ci dessus :

Le premier élément est le document encapsulé dans l'interface Document

Toutes les classes et interfaces sont regroupées dans le package org.w3c.dom

33.6.1. Les interfaces du DOM

Chaque type d'entité qui compose l'arbre est défini dans une interface. L'interface de base est l'interface Node. Plusieurs autres interfaces héritent de cette interface.

33.6.1.1. L'interface Node

Chaque élément de l'arbre est un noeud encapsulé dans l'interface org.w3c.dom.Node ou dans une de ces interfaces filles.

L'interface définit plusieurs méthodes :

Méthode	Rôle
short getNodeType()	Renvoie le type du noeud
String getNodeName()	Renvoie le nom du noeud
String getNodeValue()	Renvoie la valeur du noeud
NamedNodeList getAttributes()	Renvoie la liste des attributs ou null
void setNodeValue(String)	Mettre à jour la valeur du noeud
boolean hasChildNodes()	Renvoie un booleen qui indique si le noeud à au moins un noeud fils
Node getFirstChild()	Renvoie le premier noeud fils du noeud ou null
Node getLastChild()	Renvoie le dernier noeud fils du noeud ou null
NodeList getChildNodes()	Renvoie une liste des noeuds fils du noeud ou null
Node getParentNode()	Renvoie le noeud parent du noeud ou null
Node getPreviousSibling()	Renvoie le noeud frère précédent
Node getNextSibling()	Renvoie le noeud frère suivant
Document getOwnerDocument()	Renvoie le document dans lequel le noeud est inclus
Node insertBefore(Node, Node)	Insère le premier noeud fourni en paramètre avant le second noeud
Node replaceNode(Node, Node)	Remplace le second noeud fourni en paramètre par le premier
Node removeNode(Node)	Supprime le noeud fourni en paramètre
Node appendChild(Node)	Ajout le noeud fourni en paramètre aux noeuds enfants du noeud courant
Node clodeNode(boolean)	Renvoie une copie du noeud. Le boolean fourni en paramètre indique si la copie doit inclure les noeuds enfants

Tous les différents noeuds qui composent l'arbre héritent de cette interface. La méthode getNodeType() permet de connaître le type du noeud. Le type est très important car il permet de savoir ce que contient le noeud.

Le type de noeud peut être :

Constante	Valeur	Rôle
ELEMENT_NODE	1	Element
ATTRIBUTE_NODE	2	Attribut
TEXT_NODE	3	
CDATA_SECTION_NODE	4	
ENTITY_REFERENCE_NODE	5	
ENTITY_NODE	6	
PROCESSING_INSTRUCTION_NODE	7	

COMMENT_NODE	8	
DOCUMENT_NODE	9	Racine du document
DOCUMENT_TYPE_NODE	10	
DOCUMENT_FRAGMENT_NODE	11	
NOTATION_NODE	12	

33.6.1.2. L'interface NodeList

Cette interface définit une liste ordonnée de noeuds suivant l'ordre du document XML. Elle définit deux méthodes :

Méthode	Rôle
int getLength()	Renvoie le nombre de noeuds contenus dans la liste
Node item(int)	Renvoie le noeud dont l'index est fourni en paramètre

33.6.1.3. L'interface Document

Cette interface définit les caractéristiques pour un objet qui sera la racine de l'arbre DOM. Cette interface hérite de l'interface Node.

Un objet de type Document possède toujours un type de noeud DOCUMENT_NODE.

Méthode	Rôle
DocumentType getDocType()	renvoie le nombre de noeuds contenu dans la liste
Element getDocumentElement()	renvoie l'élément racine du document
NodeList getElementsByTagNames(String)	renvoie une liste des éléments dont le nom est fourni en paramètre
Attr createAttributes(String)	créé un attribut dont le nom est fourni en paramètre
CDATASection createCDATASection(String)	créé un noeud de type CDATA
Comment createComment(String)	créé un noeud de type commentaire
Element createElement(string)	créé un noeud de type élément dont le nom est fourni en paramètre

33.6.1.4. L'interface Element

Cette interface définit des méthodes pour manipuler un élément et en particulier les attributs d'un élément. Un élement dans un document XML correspondant à un tag. L'interface Element hérite de l'interface Node.

Un objet de type Element à toujours pour type de noeud ELEMENT_NODE

Méthode	Rôle
String getAttribute(String)	renvoie la valeur de l'attribut dont le nom est fourni en paramètre
removeAttribut(String)	
setAttribut(String, String)	modifier ou créer un attribut dont le nom est fourni en premier paramètre et la valeur en second
String getTagName()	renvoie le nom du tag
Attr getAttributeNode(String)	renvoie un objet de type Attr qui encapsule l'attribut dont le nom est fourni en paramètre

Attr removeAttributeNode(Attr)	supprime l'attribut fourni en paramètre
Attr setAttributNode(Attr)	modifier ou créer un attribut
NodeList getElementsByTagName(String)	renvoie une liste des noeuds enfants dont le nom correspond au paramètre fourni

33.6.1.5. L'interface CharacterData

Cette interface défine des méthodes pour manipuler les données de type PCDATA d'un noeud.

Méthode	Rôle
appendData()	Ajouter le texte fourni en paramètre aux données courantes
getData()	Renvoie les données sous la forme d'une une chaîne de caractères
setData()	Permet d'initialiser les données avec la chaîne de caractères fournie en paramètre

33.6.1.6. L'interface Attr

Cette interface définit des méthodes pour manipuler les attributs d'un élement.

Les attributs ne sont pas des noeuds dans le modèle DOM. Pour pouvoir les manipuler, il faut utiliser un objet de type Element.

Méthode	Rôle
String getName()	renvoie le nom de l'attribut
String getValue()	renvoie la valeur de l'attribut
String setValue(String)	mettre le valeur avec celle fournie en parametre

33.6.1.7. L'interface Comment

Cette interface permet de caractériser un noeud de type commentaire.

Cette interface étend simplement l'interface CharacterData. Un objet qui implémente cette interface générera un tag de la forme <!−− −−> .

33.6.1.8. L'interface Text

Cette interface permet de caractériser un noeud de type Text. Un tel noeud représente les données d'un tag ou la valeur d'un attribut.

33.6.2. Obtenir un arbre DOM

Pour pouvoir utiliser un arbre DOM représentant un document, il faut utiliser un parseur qui implémente DOM. Ce dernier va parcourir le document XML et créer l''arbre DOM correspondant. Le but est d'obtenir un objet qui implémente l'interface Document car cet objet est le point d'entrée pour toutes opérations sur l'arbre DOM.

Avant la définition de JAXP par Sun, l'instanciation d'un parseur était spécifique à chaque à implémentation.

Exemple : utilisation de Xerces sans JAXP

```
package perso.jmd.tests.testdom;

import org.apache.xerces.parsers.*;
import org.w3c.dom.*;

public class TestDOM2 {

  public static void main(String[] args) {
    Document document = null;
    DOMParser parser = null;

    try {
      parser = new DOMParser();
      parser.parse("test.xml");
      document = parser.getDocument();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

JAXP permet, si le parseur respecte ces spécifications, d'instancier le parseur de façon normalisée.

Exemple : utilisation de Xerces avec JAXP

```
package perso.jmd.tests.testdom;

import org.w3c.dom.*;
import javax.xml.parsers.*;

public class TestDOM1 {

  public static void main(String[] args) {
    Document document = null;
    DocumentBuilderFactory factory = null;

    try {
      factory = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = factory.newDocumentBuilder();
      document = builder.parse("test.xml");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

L'utilisation de JAXP est fortement recommandée.

Remarque : JAXP est détaillé dans une des sections suivantes de ce chapitre.

33.6.3. Parcours d'un arbre DOM

Cette section sera développée dans une version future de ce document

33.6.3.1. Les interfaces Traversal

DOM level 2 propose plusieurs interfaces pour faciliter le parcours d'un arbre DOM.

La suite de cette section sera développée dans une version future de ce document

33.6.4. Modifier un arbre DOM

Un des grands intérêts du DOM est sa faculté à créer ou modifier l'arbre qui représente un document XML.

33.6.4.1. La création d'un document

La méthode newDocument() de la classe DocumentBuilder renvoie une nouvelle instance d'un objet de type document qui encapsule un arbre DOM vide.

Il faut a minima ajouter un tag racine au document XML. Pour cela, il faut appeler la méthode createElement() de l'objet Document en lui passant le nom du tag racine pour obtenir une référence sur le nouveau noeud. Il suffit ensuite d'utiliser la méthode appendChild() de l'objet Document en lui fournissant la référence sur le noeud en paramètre.

Exemple :

```
package perso.jmd.tests.testdom;

import org.w3c.dom.*;
import javax.xml.parsers.*;

public class TestDOM09 {

  public static void main(String[] args) {
    Document document = null;
    DocumentBuilderFactory fabrique = null;

    try {
      fabrique = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = fabrique.newDocumentBuilder();
      document = builder.newDocument();
      Element racine = (Element) document.createElement("bibliotheque");
      document.appendChild(racine);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

33.6.4.2. L'ajout d'un élément

L'interface Document propose plusieurs méthodes createXXX pour créer des instances de différents types d'éléments. Il suffit alors d'utiliser la méthode appendChild() d'un noeud pour lui attacher un noeud fils.

Exemple :

```
    Element monElement = document.createElement("monelement");
     Element monElementFils = document.createElement("monelementfils");
     monElement.appendChild(monElementFils);
```

Pour ajouter un texte à un noeud, il faut utiliser la méthode createTextNode() pour créer un noeud de type Text et l'ajouter au noeud concerné avec la méthode appendChild().

Exemple :

```
    Element monElementFils = document.createElement("monelementfils");
     monElementFils.appendChild(document.createTextNode("texte du tag fils");
     monElement.appendChild(monElementFils);
```

Pour ajouter un attribut à un élément, il existe deux méthodes : setAttributeNode() et setAtribute().

La méthode setAttributeNode() attend un objet de type Attr qu'il faut préalablement instancier.

Exemple :

```
    Attr monAttribut = document.createAttribute("attribut");
     monAttribut.setValue("valeur");
     monElement.setAttributeNode(monAttribut);
```

La méthode setAttribut permet directement d'associer un attribut en lui fournissant en paramètre son nom et sa valeur.

Exemple :

```
   monElement.setAttribut("attribut","valeur");
```

La création d'un commentaire se fait en utilisant la méthode createComment() de la classe Document.

Toutes ces actions permettent la création complète d'un arbre DOM représentant un document XML.

Exemple : un exemple complet

```
package perso.jmd.tests.testdom;

import org.w3c.dom.*;
import javax.xml.parsers.*;

public class TestDOM11 {

  public static void main(String[] args) {
    Document document = null;
    DocumentBuilderFactory fabrique = null;

    try {
      fabrique = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = fabrique.newDocumentBuilder();
      document = builder.newDocument();
      Element racine = (Element) document.createElement("bibliotheque");
      document.appendChild(racine);
      Element livre = (Element) document.createElement("livre");
      livre.setAttribute("style", "1");
      Attr attribut = document.createAttribute("type");
      attribut.setValue("broche");
      livre.setAttributeNode(attribut);
      racine.appendChild(livre);
      livre.setAttribute("style", "1");
      Element titre = (Element) document.createElement("titre");
      titre.appendChild(document.createTextNode("Titre 1"));
      livre.appendChild(titre);
      racine.appendChild(document.createComment("mon commentaire"));
```

```
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Résultat :

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliotheque>
    <livre style="1" type="broche">
        <titre>Titre 1</titre>
    </livre>
    <!--mon commentaire-->
</bibliotheque>
```

33.6.5. Envoyer un arbre DOM dans un flux

Une fois un arbre DOM créé ou modifié, il est souvent utile de l'envoyer dans un flux (sauvegarde dans un fichier ou une base de données, envoi dans un message JMS ...).

Bizarrement, DOM level 1 et 2 ne propose rien pour réaliser cette tache quasiment obligatoire à effectuer. Ainsi, chaque implémentation propose sa propre méthode en attendant des spécifications qui feront surement partie du DOM Level 3.

33.6.5.1. Exemple avec Xerces

Xerces fournit la classe XMLSerializer qui permet de créer un document XML à partir d'un arbre DOM.

Xerces est téléchargeable sur le site web http://xml.apache.org/xerces2-j/ sous la forme d'une archive de type zip qu'il faut décompresser dans un répertoire du système. Il suffit alors d'ajouter les fichier xmlParserAPIs.jar et xercesImpl.jar dans le classpath.

Exemple :

```
package perso.jmd.tests.testdom;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.apache.xml.serialize.*;

public class TestDOM10 {

  public static void main(String[] args) {
    Document document = null;
    DocumentBuilderFactory fabrique = null;

    try {
      fabrique = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = fabrique.newDocumentBuilder();
      document = builder.newDocument();
      Element racine = (Element) document.createElement("bibliotheque");
      document.appendChild(racine);

      for (int i = 1; i < 4; i++) {
        Element livre = (Element) document.createElement("livre");
        Element titre = (Element) document.createElement("titre");
        titre.appendChild(document.createTextNode("Titre "+i));
        livre.appendChild(titre);
        Element auteur = (Element) document.createElement("auteur");
        auteur.appendChild(document.createTextNode("Auteur "+i));
        livre.appendChild(auteur);
        Element editeur = (Element) document.createElement("editeur");
        editeur.appendChild(document.createTextNode("Editeur "+i));
        livre.appendChild(editeur);
```

```
        racine.appendChild(livre);
      }

      XMLSerializer ser = new XMLSerializer(System.out, new OutputFormat("xml", "UTF-8", true));
      ser.serialize(document);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Résultat :

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliotheque>
    <livre>
        <titre>Titre 1</titre>
        <auteur>Auteur 1</auteur>
        <editeur>Editeur 1</editeur>
    </livre>
    <livre>
        <titre>Titre 2</titre>
        <auteur>Auteur 2</auteur>
        <editeur>Editeur 2</editeur>
    </livre>
    <livre>
        <titre>Titre 3</titre>
        <auteur>Auteur 3</auteur>
        <editeur>Editeur 3</editeur>
    </livre>
</bibliotheque>
```

33.7. La génération de données au format XML

Il existe plusieurs façon de générer des données au format XML :

- coder cette génération à la main en écrivant dans un flux

Exemple :

```
public void service(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException {
  response.setContentType("text/xml");
  PrintWriter out = response.getWriter();

  out.println("<?xml version=\"1.0\"?>");
  out.println("<BIBLIOTHEQUE>");
  out.println(" <LIVRE>");
  out.println(" <TITRE>titre livre 1</TITRE>");
  out.println(" <AUTEUR>auteur 1</AUTEUR>");
  out.println(" <EDITEUR>editeur 1</EDITEUR>");
  out.println(" </LIVRE>");
  out.println(" <LIVRE>");
  out.println(" <TITRE>titre livre 2</TITRE>" );
  out.println(" <AUTEUR>auteur 2</AUTEUR>");
  out.println(" <EDITEUR>editeur 2</EDITEUR>" );
  out.println(" </LIVRE>");
  out.println(" <LIVRE>");
  out.println(" <TITRE>titre livre 3</TITRE>");
  out.println(" <AUTEUR>auteur 3</AUTEUR>");
  out.println(" <EDITEUR>editeur 3</EDITEUR>");
  out.println(" </LIVRE>");
  out.println("</BIBLIOTHEQUE>");
  }
}
```

- utiliser JDOM pour construire le document et le sauvegarder

La suite de cette section sera développée dans une version future de ce document

33.8. JAXP : Java API for XML Parsing

JAXP est une API développée par Sun qui ne fournit pas une nouvelle méthode pour parser un document XML mais propose une interface commune pour appeler et paramétrer un parseur de façon indépendante de tout fournisseur et normaliser la source XML à traiter. En utilisant un code qui respecte JAXP, il est possible d'utiliser n'importe quel parseur qui répond à cette API tel que Crimson le parseur de Sun ou Xerces le parseur du groupe Apache.

JAXP supporte pour le moment les parseurs de type SAX et DOM.

	JAXP 1.0	JAXP 1.1
SAX	type 1	type 2
DOM	niveau 1	niveau 2

Par exemple, sans utiliser JAXP, il existe deux méthodes pour instancier un parseur de type SAX :

- créer une instance de la classe de type SaxParser
- utiliser la classe ParserFactory qui demande en paramètre le nom de la classe de type SaxParser

Ces deux possibilités nécessitent une recompilation d'une partie du code lors du changement du parseur.

JAXP propose de fournir le nom de la classe du parseur en paramètre à la JVM sous la forme d'une propriété système. Il n'est ainsi plus nécessaire de procéder à une recompilation mais simplement de mettre jour cette propriété et le CLASSPATH pour qu'il référence les classes du nouveau parseur.

Le parseur de Sun et les principaux parseurs XML en Java implantent cette API et il est très probable que tous les autres fournisseurs suivent cet exemple.

33.8.1. JAXP 1.1

JAXP version 1.1 contient une documentation au format javadoc, des exemples et trois fichiers jar :

- jaxp.jar : contient l'API JAXP
- crimson.jar : contient le parseur de Sun
- xalan.jar : contient l'outil du groupe apache pour les transformations XSL

L'API JAXP est fournie avec une implémentation de référence de deux parseurs (une de type SAX et une de type DOM) dans le package org.apache.crimson et ses sous packages.

JAXP ce compose de plusieurs packages :

- javax.xml.parsers
- javax.xml.transform
- org.w3c.dom
- org.xml.sax

JAXP définit deux exceptions particulières :

- FactoryConfigurationError est levée si la classe du parseur précisée dans la variable System ne peut être instanciée
- ParserConfigurationException est levée lorsque les options précisées dans la factory ne sont pas supportées par le parseur

33.8.2. L'utilisation de JAXP avec un parseur de type SAX

L'API JAXP fournie la classe abstraite SAXParserFactory qui fournie une méthode pour récupérer une instance d'un parseur de type SAX grace à une de ces méthodes. Une classe fille de la classe SAXParserFactory permet d'être instanciée.

La propriété système javax.xml.parsers.SAXParserFActory permet de préciser la classe fille qui hérite de la classe SAXParserFactory et qui sera instanciée.

Remarque : cette classe n'est pas thread safe.

La méthode statique newInstance() permet d'obtenir une instance de la classe SAXParserFactory : elle peut lever une exception de type FactoryConfigurationError.

Avant d'obtenir une instance du parseur, il est possible de fournir quelques paramètres à la Factory pour lui permettre de configurer le parseur.

La méthode newSAXParser() permet d'obtenir une instance du parseur de type SAXParser : peut lever une exception de type ParserConfigurationException.

Les principales méthodes sont :

Méthode	Rôle
boolean isNamespaceAware()	indique si la factory est configurée pour instancier des parseurs qui prennent en charge les espaces de noms
boolean isValidating()	indique si la factory est configurée pour instancier des parseurs qui valide le document XML lors de son traitement
static SAXParserFactory newInstance()	permet d'obtenir une instance de la factory
SAXParser newSAXParser()	permet d'obtenir une nouvelle instance du parseur de type SAX configuré avec les options fournie à la factory
setNamespaceAware(boolean)	configure la factory pour instancier un parseur qui prend en charge les espaces de noms ou non selon le paramètre fourni
setValidating(boolean)	configure la factory pour instancier un parseur qui valide le document XML lors de son traitement ou non selon le paramètre fourni

Exemple :

```
SAXParserFActory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
parser.parse(new File(args[0), new handler());
```

33.9. XSLT (Extensible Stylesheet Language Transformations)

XSLT est une recommandation du consortium W3C qui permet de transformer facilement des documents XML en d'autres documents standard sans programmation. Le principe est de définir une feuille de style qui indique comment transformer le document XML et de le fournir avec le document à un processeur XSLT.

On peut produire des documents de différents formats : XML, HTML, XHTML, WML, PDF, etc...

XSLT fait parti de XSL avec les recommandations :

- XSL−FO : flow object
- XPath : langage pour spécifier un élement dans un document. Ce langage est utilisé par XSL.

Une feuille de style XSLT est un fichier au format XML qui contient les informations nécessaires au processeur pour effectuer la transformation.

Le composant principal d'une feuille de style XSLT est le template qui définit le moyen de transformer un élément du document XML dans le nouveau document.

XSLT est très relativement complet et complexe : cette section n'est qu'une présentation rapide de quelques fonctionnalités de XSLT. XSLT possède plusieurs fonctionnalités avancées, tel que la sélection des éléments à traiter, filter des éléments ou trier les éléments.

33.9.1. XPath

XML Path ou XPath est une spécification qui fournit une syntaxe pour permettre de sélectionner un ou plusieurs éléments dans un document XML. Il existe sept types d'éléments différents :

- racine (root)
- element
- text
- attribute (attribute)
- commentaire (comment)
- instruction de traitement (processing instruction)
- espace de nommage (name space)

Cette section ne présente que les fonctionnalités de base de XPath.

XPath est utilisé dans plusieurs technologies liées à XML tel que XPointer et XSLT.

Un document XMLpeut être représenté sous la forme d'un arbre composé de noeud. XPath grace à une notation particulière permet localisation précisement un composant de l'arbre.

La notation reprend une partie de la notation utiliser pour naviguer dans un système d'exploitation, ainsi :

- le séparateur est le le slash /
- pour préciser un chemin à partir de la racine (chemin absolue), il faut qu'il commence par un /
- un double point .. permet de préciser l'élément père de l'élément courant
- un simple point . permet de préciser l'élément courant
- un arobase @ permet de préciser un attribut d'un élément
- pour préciser l'indice d'un élément il faut le préciser entre crochets

XPath permet de filtrer les élements sur différents critères en plus de leur nom

- @categorie="test" : recherche un attribut dont le nom est categorie est dont la valeur est "test"
- une barre verticale | permet de préciser deux valeurs

33.9.2. La syntaxe de XSLT

Une feuille de style XSLT est un document au format XML. Il doit donc respecter toute les règles d'un tel document. Pour préciser les différentes instructions permettant de réaliser la transformation, un espace de nommage particulier est

utilisé : xsl. Tout les tags de XSLT commencent donc par ce préfixe, ainsi le tag racine du document est xsl:stylesheet. Ce tag racine doit obligatoirement posséder un attribut version qui précise la version de XSLT utilisé.

Exemple : une feuille de style minimale qui ne fait rien

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>
```

Le tag xsl:output permet de préciser le format de sortie. Ce tag possède plusieurs attributs :

- method : cet attribut permet de préciser le format. Les valeurs possibles sont : texte, xml ou html
- indent : cet attribut permet de définir si la sortie doit être indentée ou non. Les valeurs possiblent sont : yes ou no
- encoding : cet attribut permet de préciser le jeux de caractères utilisé pour la sortie

Pour effectuer la transformation, le document doit contenir des règles. Ces règles suivent une syntaxe particulière et sont contenues dans des modèles (templates) associés à un ou plusieurs éléments désignés avec un motif au format XPath.

Un modèle est défini grace au tag xsl:template. La valeur de l'attribut match permet de fournir le motif au format XPath qui sélectionnera le ou les élements sur lequel le modèle va agir.

Le tag xsl:apply−templates permet de demander le traitements des autres modèles définis pour chacun des noeuds fils du noeud courant.

Le tag xsl:value−of permet d'extraire la valeur de l'élément respectant le motif XPath fourni avec l'attribut select.

Il existe beaucoup d'autre tags notamment plusieurs qui permettent d'utiliser de structures de contrôles de type itératifs ou conditionnels.

Le tag xsl:for−each permet parcourir un ensemble d'élement sélectionner par l'attribut select. Le modèle sera appliqué sur chacun des éléments de la liste

Le tag xsl:if permet d'exécuter le modèle si la condition précisée par l'attribut test au format XPath est juste. XSLT ne défini pas de tag équivalent à la partie else : il faut définir un autre tag xsl:if avec une condition opposée.

Le tag xsl:choose permet de définir plusieurs conditions. Chaque condition est précisée grace à l'attribut xsl:when avec l'attribut test. Le tag xsl:otherwise permet de définir un cas par défaut qui ne correspond aux autres cas défini dans le tag xsl:choose.

Le tag xsl:sort permet de trier un ensemble d'éléments. L'attribut select permet de préciser les élément qui doivent être triés. L'attribut data−type permet de préciser le format des données (text ou number). l'attribut order permet de préciser l'ordre de tri (ascending ou descending).

33.9.3. Exemple avec Internet Explorer

Le plus simple pour tester une feuille XSLT qui génère une page HTML est de la tester avec Internet Explorer version 6. Cette version est entièrement compatible avec XML et XSLT. Les versions 5 et 5.5 ne sont que partiellement compatibles. Les versions antérieures ne le sont pas du tout.

33.9.4. Exemple avec Xalan 2

Xalan 2 utilise l'API JAXP.

Exemple : TestXSL2.java

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import java.io.IOException;

public class TestXSL2
{
  public static void main(String[] args)
        throws TransformerException, TransformerConfigurationException,
         SAXException, IOException
        {

    TransformerFactory tFactory = TransformerFactory.newInstance();
    Transformer transformer = tFactory.newTransformer(new StreamSource("test.xsl"));

    transformer.transform(new StreamSource("test.xml"), new StreamResult("test.htm"));
  }
}
```

Exemple : la feuille de style XSL test.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/REC-html40">

<xsl:output method="html" indent="no" />
<xsl:strip-space elements="*" />

<xsl:template match="/">
<HTML>
<HEAD>
<TITLE>Test avec XSL</TITLE>
</HEAD>
<xsl:apply-templates />
</HTML>
</xsl:template>

<xsl:template match="BIBLIOTHEQUE">
<BODY>
<H1>Liste des livres</H1>
<TABLE border="1" cellpadding="4">
<TR><TD>Titre</TD><TD>Auteur</TD><TD>Editeur</TD></TR>
<xsl:apply-templates />
</TABLE>
</BODY>
</xsl:template>

<xsl:template match="LIVRE">
<TR>
<TD><xsl:apply-templates select="TITRE"/></TD>
<TD><xsl:apply-templates select="AUTEUR"/></TD>
<TD><xsl:apply-templates select="EDITEUR"/></TD>
</TR>
</xsl:template>
</xsl:stylesheet>
```

Exemple : compilation et execution avec Xalan

```
javac TestXSL2.java -classpath .;xalan2.jar
java -cp .;xalan2.jar TestXSL2
```

La suite de cette section sera développée dans une version future de ce document

33.10. Les modèles de document

Devant les faibles possibilités de manipulation d'un document XML avec SAX et le manque d'intégration à java de DOM (qui n'a pas été écrit pour java), plusieurs projets se sont développés pour proposer des modèles de représentation des documents XML qui utilise un ensemble d'api se basant sur celles existant en java.

Les deux projets les plus connus sont JDOM et Dom4J. Bien que différents, ces deux projets ont de nombreux points communs :

- ils ont le même but
- ils sont écrits en java et ne s'utilisent qu'en java
- ils utilisent soit SAX soit DOM pour créer le modèle de document en mémoire
- ils utilisent l'api Collection de Java2.

33.11. JDOM

Malgrès la similitude de nom entre JDOM et DOM, ces deux API sont très différentes. JDOM est une API uniquement java car elle s'appuie sur un ensemble de classe de java notamment les collections. Le but de JDOM n'est pas de définir un nouveau type de parseur mais de faciliter la manipulation au sens large de document XML : lecture d'un document, représentation sous forme d'arborescence, manipulation de cet arbre, définition d'un nouveau document, exportation vers plusieurs cibles ... Dans le rôle de manipulation sous forme d'arbre, JDOM possède moins de fonctionnalités que DOM mais en contre partie il offre une plus grande facilité pour répondre au cas les plus classiques d'utilisation.

Pour parser un document XML, JDOM utilise un parseur externe de type SAX ou DOM.

JDOM est une Java Specification Request numéro 102 (JSR−102).

33.11.1. Installation de JDOM sous Windows

Pour utiliser JDOM il faut construire la bibliothèque grace à l'outil ant. Ant doit donc être installé sur la machine.

Il faut aussi que la variable JAVA_HOME soit positionnée avec le répertoire qui contient le JDK.

```
set JAVA_HOME=c:\j2sdk1.4.0−rc
```

Il suffit alors simplement d'executer le fichier build.bat situé dans le répertoire d'installation de jdom.

Un message informe de la fin de la compilation :

```
package:
[jar] Building jar: C:\java\jdom−b7\build\jdom.jar

BUILD SUCCESSFUL

Total time: 1 minutes 33 seconds
```

Le fichier jdom.jar est créé dans le répertoire build.

Pour utiliser JDOM dans un projet, il faut obligatoirement avoir un parseur XML SAX et/ou DOM. Pour les exemples de cette section j'ai utilisé Xerces. Il faut aussi avoir le fichier jaxp.jar.

Pour compiler et éxecuter les exemples de cette section, j'ai utilisé le script suivant :

```
javac %1.java −classpath .;jdom.jar;xerces.jar;jaxp.jar

java −classpath .;jdom.jar;xerces.jar;jaxp.jar %1
```

33.11.2. Les différentes entités de JDOM

Pour traiter un document XML, JDOM définit plusieurs entités qui peuvent être regroupées en trois groupes :

- les éléments de l'arbre
  - ♦ le document : la classe Document
  - ♦ les éléments : la classe Element
  - ♦ les commentaires : la classe Comment
  - ♦ les attributs : la classe Attribute
- les entités pour obtenir un parseur :
  - ♦ les classe SAXBuilder et DOMBuilder
- les entités pour produire un document
  - ♦ les classes XMLOutputter, SAXOutputter, DOMOutputter

Ces classes sont regroupées dans cinq packages :

- org.jdom
- org.jdom.adapters
- org.jdom.input
- org.jdom.output
- org.jdom.transform

Attention : cette API a énormément évoluée jusqu'à sa version 1.0. Beaucoup de méthodes ont été déclarées deprecated.

33.11.3. La classe Document

La classe org.jdom.Document encapsule l'arbre dans le lequel JDOM stocke le document XML.Pour obtenir un objet Document, il y a deux possibilités :

- utiliser un objet XXXBuilder qui va parser un document XML existant et créer l'objet Document en utilisant un parseur
- instancier un nouvel objet Document pour créer un nouveau document XML

Pour créer un nouveau document, il suffit d'instancier un objet Document en utilisant un des constructeurs fournis dont les principaux sont :

Constructeur	Rôle
Document()	
Document(Element)	Création d'un document avec l'élement racine fourni
Document(Element, DocType)	Création d'un document avec l'élement racine et la déclaration doctype fourni

Exemple (code Java 1.2) :

```
import org.jdom.*;

public class TestJDOM2 {
        public static void main(String[] args) {
                Element racine = new Element("bibliothèque");
                Document document = new Document(racine);

        }
}
```

Pour obtenir un document à partir d'un document XML existant, JDOM propose deux classes regroupées dans le package org.jdom.input qui implémentent l'interface Builder. Cette interface définit la méthode build() qui renvoie un objet de type Document et qui est surchargée pour utiliser trois sources différentes : InputStream, File et URL. Les deux classes sont SAXBuilder et DOMbuilder.

La classe SAXBuilder permet de parser le document XML avec un parseur de type SAX compatible JAXP, de créer un arbre JDOM et renvoie un objet de type Document. SAXBuilder possède plusieurs constructeurs qui permettent de préciser la classe du parseur à utiliser et/ou un boolean qui indique si le document doit être validé.

Exemple (code Java 1.2) :

```
import org.jdom.*;
import org.jdom.input.*;
import java.io.*;

public class TestJDOM3 {
        public static void main(String[] args) {
                try {
                        SAXBuilder builder = new SAXBuilder();
                        Document document = builder.build(new File("test.xml"));
                } catch(JDOMException e) {
                        e.printStackTrace();
                }

        }
}
```

La classe Document possède plusieurs méthodes dont les principales sont :

Méthode	Rôle
Document addContent(Comment)	Ajouter un commentaire au document
List getContent()	Renvoie un objet List qui contient chaque élément du document
DocType getDocType()	Renvoie un objet contenant les caractéristiques doctype du document
Element getRootElement()	Renvoie l'élément racine du document
Document setRootElement(Element)	Défini l'élément racine du document

La classe DocType encapsule les données sur le type du document.

33.11.4. La classe Element

La classe Element représente un élément du document. Un élément peut contenir du texte, des attributs, des commentaires, et tous les autres éléments définis par la norme XML. Cette classe possède plusieurs constructeurs dont les principaux sont :

Constructeur	Rôle
Element()	Créer un objet par défaut
Element(String)	Créer un objet, en précisant son nom
Element(String, String)	Créer un objet, en précisant son nom et son espace de nommage

La classe Element possède de nombreuses méthodes pour obtenir, ajouter ou supprimer une entité de l'élément (une élement enfant, un texte, un attribut, un commentaire ...).

Méthode	Rôle
Element addContent(Comment)	Ajouter un commentaire à l'élément
Element addContent(Element)	Ajouter un élément fils à l'élément
Element addContent(String text)	Ajouter des données sous forme de texte à l'élément
Attribute getAttribute(String)	Renvoie l'attribut dont le nom est fourni en paramètre
Attribute getAttribute(String, Namespace)	Renvoie l'attribut dont le nom et l'espace de nommage sont fournis en paramètre
List getAttributes()	Renvoie une liste qui contient tous les attributs
String getAttributeValue(String)	Renvoie la valeur de l'attribut dont le nom est fourni en paramètre
String getAttributeValue(String, Namespace)	Renvoie la valeur de l'attribut dont le nom et l'espace de nommage sont fourni en paramètre
Element getChild(String)	Renvoie le premier élément enfant dont le nom est fourni en paramètre
Element getChild(String, Namespace)	Renvoie le premier élément enfant dont le nom et l'espace de nommage sont fourni en paramètre
List getChildren()	Renvoie une liste qui contient tous les élements enfants directement rattachés à l'élément
List getChildren(String)	Renvoie une liste qui contient tous les élements enfants directement rattachés à l'élément dont le nom est fourni en paramètre
List getChildren(String, Namespace)	Renvoie une liste qui contient tous les élements enfants directement rattachés à l'élément dont le nom et l'espace de nommage est fourni en paramètre
String getChildText(String name)	Renvoie le texte du premier élément enfant dont le nom est fourni en paramètre
String getChildText(String, Namespace)	Renvoie le texte du premier élément enfant dont le nom et l'espace de nommage sont fourni en paramètre
List getContent()	Renvoie une liste qui contient contient toutes les entités de l'élément (texte, element, commentaire ...)
Document getDocument()	Renvoie l'objet Document qui contient l'élément
Element getParent()	Renvoie l'élément père de l'élément
String getText()	Renvoie les données au format texte contenues dans l'élément
boolean hasChildren()	Renvoie un élément qui indique si l'lément possède des élements fils
boolean isRootElement()	Renvoie un boolen qui indique si l'élément est l'élément racine du document
boolean removeAttribute(String)	Supprime l'attribut dont le nom est fourni en parametre
boolean removeAttribute(String, Namespace)	Supprime l'attribut dont le nom et l'espace de nommage sont fournis en paramètre
boolean removeChild(String)	Supprime l'élément enfant dont le nom est fourni en paramètre
boolean removeChild(String, Namespace)	Supprime l'élément enfant dont le nom et l'espace de nommage sont fournis en paramètre
boolean removeChildren()	Supprime tous les éléments enfants
boolean removeChildren(String)	Supprime tous les éléments enfants dont le nom est fourni en paramètre
boolean removeChildren(String, Namespace)	Supprime tous les éléments enfants dont le nom et l'espace de nommage sont fournis en paramètre
boolean removeContent(Comment)	Supprime le commentaire fourni en paramètre
boolean removeContent(Element)	Supprime l'élément fourni en paramètre
Element setAttribute(Attribute)	Ajoute un attribut

Element setAttribute(String, String)	Ajoute un attribut dont le nom et la valeur sont fournis en paramètre
Element setAttribute(String, String, Namespace)	Ajoute un attribut dont le nom, la valeur et l'espace de nommage sont fournis en paramètre
Element setText(String)	Mettre à jour les données au format texte de l'élément

Pour obtenir l'élément racine d'un document, il faut utiliser la méthode getRootElement() de la classe Document. Celle ci renvoie un objet de type Element. Il est ainsi possible d'utiliser les méthodes ci dessous pour parcourir et modifier le contenu du document.

L'utilisation de la classe Elément est très facile.

Pour créer un élément, il suffit d'instancier un objet de type Element.

Exemple (code Java 1.2) :

```
  Element element = new Element("element1");
          element.setAttribute("attribut1","valeur1");
          element.setAttribute("attribut2","valeur2");
```

La classe possède plusieurs méthodes pour obtenir les entités de l'élément, un élément fils particulier ou une liste d'élément fils. Un appel successif à ces méthodes permet d'obtenir un élément précis du document.

Exemple (code Java 1.2) :

```
import org.jdom.*;
import org.jdom.input.*;
import java.io.*;
import java.util.*;

public class TestJDOM5 {
  public static void main(String[] args) {
    try {
      SAXBuilder builder = new SAXBuilder();
      Document document = builder.build(new File("test.xml"));

      Element element = document.getRootElement();
      List livres=element.getChildren("LIVRE");
      ListIterator iterator = livres.listIterator();
      while (iterator.hasNext()) {
        Element el = (Element) iterator.next();
        System.out.println("titre = "+el.getChild("TITRE").getText());
      }
    } catch(JDOMException e) {
      e.printStackTrace(System.out);
    }
  }
}
```

Résultat:

```
titre = titre livre 1
titre = titre livre 2
titre = titre livre 3
```

L'inconvénient d'utiliser un nom de tag pour rechecher un élément est qu'il faut être sur que l'élément existe sinon une exception de type nullPointerException est levée. Le plus simple est d'avoir une DTD et d'activer la validation du document pour le parseur.

33.11.5. La classe Comment

Cette section sera développée dans une version future de ce document

33.11.6. La classe Namespace

JDOM gère les espace de nom grace à la classe Namespace. Un espace de nom est composé d'un préfix auquel on associé une URI. JDOM gére les espaces de nom de lui même. La méthode statique getNamespace() permet de retrouver ou de créer un espace de nom.

33.11.7. La classe Attribut

Cette section sera développée dans une version future de ce document

33.11.8. La sortie de document

JDOM prévoit plusieurs classes pour permettre d'exporter le document contenu dans un objet de type Document.

Le plus utilisé est de type XMLOutputter qui permet d'envoyer le document XML dans un flux. Il est possible de fournir plusieurs paramètres pour formatter la sortie du document. Cette classe possède plusieurs constructeurs dont les principaux sont :

Constructeur	Rôle
XMLOutputter()	Créer un objet par défaut, sans paramètre de formattage
XMLOutputter(String)	Créer un objet, en précisant une chaîne pour l'indentation
XMLOutputter(String, boolean)	Créer un objet, en précisant une chaîne pour l'indentation et un boolean qui indique si une nouvelle ligne doit être ajoutée après chaque élément
XMLOutputter(String, boolean, String)	Créer un objet, en précisant une chaîne pour l'indentation, un boolean qui indique si une nouvelle ligne doit être ajoutée après chaque élément et une chaîne qui précise le jeux de caractères à utiliser pour formatter le document

```
XMLOutputter outputter = new XMLOutputter();
outputter.output(doc, System.out);
```

Si le fichier XML n'a besoin d'être exploité que par un programme informatique, il est préférable de supprimer toute forme de formattage.

```
XMLOutputter outputter = new XMLOutputter("", false);
outputter.output(doc, System.out);
```

La suite de cete section sera développée dans une version future de ce document

33.12. dom4j

dom4j est un framework open source pour manipuler des données XML, XSL et Xpath. Il est entièrement développé en Java et pour Java.

Dom4j n'est pas un parser mais propose un modèle de représentation d'un document XML et une API pour en faciliter l'utilisation. Pour obtenir une telle représentation, dom4j utilise soit SAX, soit DOM. Comme il est compatible JAXP, il est possible d'utiliser toute implémentation de parser qui implémente cette API.

La version de dom4j utilisée dans cette section est la 1.3

33.12.1. Installation de dom4j

Download de la dernière version à l'url http://dom4j.org/download.html

Il suffit de dezipper le fichier downloadé. Celui ci contient de nombreuses bibliothèques (ant, xalan, xerces, crimson, junit, ...), le code source du projet, et la documentation.

Le plus simple pour utiliser rapidement dom4j est de copier les fichiers jar contenus dans le répertoire lib dans le répertoire ext du répertoire %JAVA_HOME%/jre/lib/ext ainsi que les fichier dom4j.jar et dom4j−full.jar.

33.12.2. La création d'un document

Dom4j encapsule un document dans un objet de type org.dom4j.Document. Dom4j propose des API pour facilement créer un tel objet qui va être le point d'entrée de la représentation d'un document XML .

Exemple utilisant SAX :

```
import org.dom4j.*;
import org.dom4j.io.*;
public class Testdom4j_1 {
  public static void main(String args[]) {
    DOCUMENT DOCUMENT;
    try {
      SAXReader xmlReader = new SAXReader();
      document = xmlReader.read("test.xml");
    } catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

Pour éxecuter ce code, il suffit d'executer

java −cp .;%JAVA_HOME%/jre/lib/ext/dom4j−full.jar Testdom4j_1.bat

Exemple à partir d'une chaîne de caractères :

```
import org.dom4j.*;
public class Testdom4j_8 {
  public static void main(String args[]) {
    Document document = null;
    String texte = "<bibliotheque><livre><titre>titre 1</titre><auteur>auteur 1</auteur>"
         +"<editeur>editeur 1</editeur></livre></bibliotheque>";
    try {
      document = DocumentHelper.parseText(texte);
    } catch (Exception e){
     e.printStackTrace();
    }
  }
}
```

33.12.3. Le parcours d'un document

Le parcours du document construit peut se faire de plusieurs façon :

- utilisation de l'API collection
- utilisation de XPath
- utilisation du pattern Visitor

Le parcours peut se faire en utilisant l'API collection de Java.

Exemple : obtenir tous les noeuds fils du noeud racine

```
import org.dom4j.*;
import org.dom4j.io.*;
import java.util.*;
public class Testdom4j_2 {
  public static void main(String args[]) {
    Document document;
    org.dom4j.Element racine;
    try {
      SAXReader xmlReader = new SAXReader();
      document = xmlReader.read("test.xml");
      racine = document.getRootElement();
      Iterator it = racine.elementIterator();
      while(it.hasNext()){
        Element element = (Element)it.next();
        System.out.println(element.getName());
      }
    } catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

Un des grands intérêts de dom4j est de proposer une recherche dans le document en utilisant la technologie Xpath.

Exemple : obtenir tous les noeuds fils du noeud racine

```
import org.dom4j.*;
import org.dom4j.io.*;
import java.util.*;
public class Testdom4j_3 {
  public static void main(String args[]) {
    Document document;
    try {
      SAXReader xmlReader = new SAXReader();
      document = xmlReader.read("test.xml");
      XPath xpathSelector = DocumentHelper.createXPath("/bibliotheque/livre/auteur");
      List liste = xpathSelector.selectNodes(document);
      for ( Iterator it = liste.iterator(); it.hasNext(); ) {
        Element element = (Element) it.next();
```

```
        System.out.println(element.getName()+" : "+element.getText());
      }
    } catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

33.12.4. La modification d'un document XML

L'interface Document propose plusieurs méthodes pour modifier la structure du document.

Méthode	Rôle
Document addComment(String)	Ajouter un commentaire
void setDocType(DocumentType)	Modifier les caractéristiques du type de document
void setRootElement(Element)	Modifier l'élement racine du document

L'interface Element propose plusieurs méthodes pour modifier un élément du document.

Méthode	Rôle
void add(...)	Méthode surchargée qui permet d'ajouter un attribut, une entité, un espace nommage ou du texte à l'élément
Element addAttribute(String, String)	Ajouter un attribut à l'élément
Element addComment(String)	Ajouter un commentaire à l'élément
Element addEntity(String, String)	Ajouter une entité à l'élément
Element addNameSpace(String, String)	Ajouter un espace de nommage à l'élément
Element addText(String)	Ajouter un text à l'élément

33.12.5. La création d'un nouveau document XML

Il est très facile de créer un document XML

La classe DocumentHelper propose une méthode createDocument() qui renvoie une nouvelle instance de la classe Document. Il suffit alors d'ajouter chacun des noeuds de l'arbre du nouveau document XML en utilisant la méthode addElement() de la classe Document.

Exemple :

```
import org.dom4j.*;
public class Testdom4j_4 {
  public static void main(String args[]) {
    Document document = DocumentHelper.createDocument();
    Element root = document.addElement( "bibliotheque" );
    Element livre = null;
    try {
    livre = root.addElement("livre");
    livre.addElement("titre").addText("titre 1");
    livre.addElement("auteur").addText("auteur 1");
    livre.addElement("editeur").addText("editeur 1");
    } catch (Exception e){
      e.printStackTrace();
    }
  }
```

```
}
```

33.12.6. Exporter le document

Pour écrire le document XML dans un fichier, une méthode de la classe Document permet de réaliser cette action très simplement

Exemple :

```
import org.dom4j.*;
import java.io.*;
public class Testdom4j_5 {
  public static void main(String args[]) {
    Document document = DocumentHelper.createDocument();
    Element root = document.addElement( "bibliotheque" );
    Element livre = null;
    try {
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 1");
      livre.addElement("auteur").addText("auteur 1");
      livre.addElement("editeur").addText("editeur 1");
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 2");
      livre.addElement("auteur").addText("auteur 2");
      livre.addElement("editeur").addText("editeur 2");
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 3");
      livre.addElement("auteur").addText("auteur 3");
      livre.addElement("editeur").addText("editeur 3");
      FileWriter out = new FileWriter( "test2.xml" );
      document.write( out );
      out.close();
    } catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

Pour pouvoir agir sur le formattage du document ou pour utiliser un flux différent, il faut utiliser la classe XMLWriter

Exemple :

```
import org.dom4j.*;
import org.dom4j.io.*;
import java.io.*;
public class Testdom4j_6 {
  public static void main(String args[]) {
    Document document = DocumentHelper.createDocument();
    Element root = document.addElement( "bibliotheque" );
    Element livre = null;
    try {
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 1");
      livre.addElement("auteur").addText("auteur 1");
      livre.addElement("editeur").addText("editeur 1");
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 2");
      livre.addElement("auteur").addText("auteur 2");
      livre.addElement("editeur").addText("editeur 2");
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 3");
      livre.addElement("auteur").addText("auteur 3");
      livre.addElement("editeur").addText("editeur 3");
      OutputFormat format = OutputFormat.createPrettyPrint();
      XMLWriter writer = new XMLWriter( System.out, format );
      writer.write( document );
    } catch (Exception e){
      e.printStackTrace();
```

```
    }
  }
}
```

Résultat :

```
C:\test_dom4j>java -cp .;c:\j2sdk1.4.0_01/jre/lib/ext/dom4j
-full.jar Testdom4j_6
<?xml version="1.0" encoding="UTF-8"?>
<bibliotheque>
  <livre>
    <titre>titre 1</titre>
    <auteur>auteur 1</auteur>
    <editeur>editeur 1</editeur>
  </livre>
  <livre>
    <titre>titre 2</titre>
    <auteur>auteur 2</auteur>
    <editeur>editeur 2</editeur>
  </livre>
  <livre>
    <titre>titre 3</titre>
    <auteur>auteur 3</auteur>
    <editeur>editeur 3</editeur>
  </livre>
</bibliotheque>
```

La classe OutputFormat possède une méthode createPrettyPrint() qui renvoie un objet de type OutputFormat contenant des paramètres par défaut.

Il est possible d'obtenir une chaîne de caractères à partir de tout ou partie d'un document

Exemple :

```
import org.dom4j.*;
import org.dom4j.io.*;
public class Testdom4j_7 {
  public static void main(String args[]) {
    Document document = DocumentHelper.createDocument();
    Element root = document.addElement( "bibliotheque" );
    Element livre = null;
    String texte = "";
    try {
      livre = root.addElement("livre");
      livre.addElement("titre").addText("titre 1");
      livre.addElement("auteur").addText("auteur 1");
      livre.addElement("editeur").addText("editeur 1");
      texte = document.asXML();
      System.out.println(texte);
    } catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

Resultat :

```
C:\test_dom4j>java -cp .;c:\j2sdk1.4.0_01/jre/lib/ext/dom4j
-full.jar Testdom4j_7
<?xml version="1.0" encoding="UTF-8"?>
<bibliotheque><livre><titre>titre 1</titre><auteur>auteur 1</auteur><editeur>edi
teur 1</editeur></livre></bibliotheque>
```

La suite de cette section sera développée dans une version future de ce document

33.13. Jaxen

jaxen

Jaxen est un moteur Xpath qui permet de retrouver des informations grace à Xpath dans un document XML de type dom4j ou Jdom.

C'est un projet open source qui a été intégré dans dom4j pour permettre le support de Xpath dans ce framework.

33.14. JAXB

JAXB est l'acronyme de Java Architecture for XML Binding.

Le but de l'API et des spécifications JAXB est de faciliter la manipulation d'un document XML en générant un ensemble de classes qui fournissent un niveau d'abstraction plus élevé que l'utilisation de JAXP (SAX ou DOM). Avec ces deux API, toute la logique de traitements des données contenues dans le document est à écrire.

JAXB au contraire fournit un outil qui analyse un schéma XML et génère à partir de ce dernier un ensemble de classes qui vont encapsuler les traitements de manipulation du document.

Le grand avantage est de fournir au développeur un moyen de manipuler un document XML sans connaître XML ou les technologies d'analyse. Toutes les manipulations se font au travers d'objets java.

Ces classes sont utilisées pour faire correspondre le document XML dans des instances de ces classes et vice et versa : ces opérations se nomment respectivement unmarshalling et marshalling.

L'implémentation de référence de JAXB v1.0 est fournie avec le JSWDK 1.1.

Les exemples de ce chapitre utilisent cette implémentation de référence et le fichier XML suivant :

Exemple :

```
<bibliotheque>
  <livre>
    <titre>titre 1</titre>
    <auteur>auteur 1</auteur>
    <editeur>editeur 1</editeur>
  </livre>
  <livre>
    <titre>titre 2</titre>
    <auteur>auteur 2</auteur>
    <editeur>editeur 2</editeur>
  </livre>
  <livre>
    <titre>titre 3</titre>
    <auteur>auteur 3</auteur>
    <editeur>editeur 3</editeur>
  </livre>
</bibliotheque>
```

Le schéma XML correspondant à ce fichier XML est le suivant :

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="bibliotheque">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="livre" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name="livre">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="titre"/>
        <xs:element ref="auteur"/>
        <xs:element ref="editeur"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="titre" type="xs:string"/>
  <xs:element name="auteur" type="xs:string"/>
  <xs:element name="editeur" type="xs:string"/>
</xs:schema>
```

33.14.1. La génération des classes

L'outil xjc permet d'analyser un schéma XML et de générer les interfaces et les classes qui vont permettre la manipulation d'un document XML qui respecte ce schema. Cette opération se nomme binding.

La syntaxe de cet outil est très simple :

xjc [options] shema

schema est le nom d'un fichier contenant le schéma XML.

Les principales options sont les suivantes :

- −nv : ne pas réaliser une validation strict du schéma fourni
- −d repertoire : permet de préciser le nom du répertoire qui va contenir les classes générées
- −p package : permet de préciser le nom du package qui va contenir les classes générées

Exemple :

```
C:\java\jaxb>xjc test.xsd
parsing a schema...
compiling a schema...
generated\impl\AuteurImpl.java
generated\impl\BibliothequeImpl.java
generated\impl\BibliothequeTypeImpl.java
generated\impl\EditeurImpl.java
generated\impl\LivreImpl.java
generated\impl\LivreTypeImpl.java
generated\impl\TitreImpl.java
generated\Auteur.java
generated\Bibliotheque.java
generated\BibliothequeType.java
generated\Editeur.java
generated\Livre.java
generated\LivreType.java
generated\ObjectFactory.java
generated\Titre.java
generated\bgm.ser
generated\jaxb.properties
```

L'exécution de la commande de l'exemple génère les fichiers suivants :

Generated
- Auteur.java
- bgm.ser
- Bibliotheque.java
- BibliothequeType.java
- Editeur.java
- jaxb.properties
- Livre.java
- LivreType.java
- ObjectFactory.java
- Titre.java
- generated\impl
  - AuteurImpl.java
  - BibliothequeImpl.java
  - BibliothequeTypeImpl.java
  - EditeurImpl.java
  - LivreImpl.java
  - LivreTypeImpl.java
  - TitreImpl.java

Sans précision, les fichiers générés le sont dans le répertoire "Generated".

Pour préciser un autre répertoire, il faut utiliser l'option −d :

xjc −d sources test.xsd

Les classes et interfaces sont générées dans le répertoire "sources/generated"

Si le répertoire précisé n'existe pas, une exception est levée.

Exemple :

```
C:\java\jaxb>xjc −d sources test.xsd
parsing a schema...
compiling a schema...
java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
```

Pour éviter l'utilisation du répertoire "generated", il faut préciser un package pour les entités générées en utilisant l'option −p.

Exemple :

```
C:\java\jaxb>xjc −d sources −p com.moi.test.jaxb test.xsd
parsing a schema...
compiling a schema...
com\moi\test\jaxb\Auteur.java
com\moi\test\jaxb\Bibliotheque.java
com\moi\test\jaxb\BibliothequeType.java
com\moi\test\jaxb\Editeur.java
com\moi\test\jaxb\Livre.java
com\moi\test\jaxb\LivreType.java
com\moi\test\jaxb\ObjectFactory.java
com\moi\test\jaxb\Titre.java
com\moi\test\jaxb\jaxb.properties
com\moi\test\jaxb\bgm.ser
com\moi\test\jaxb\impl\AuteurImpl.java
com\moi\test\jaxb\impl\BibliothequeImpl.java
com\moi\test\jaxb\impl\BibliothequeTypeImpl.java
com\moi\test\jaxb\impl\EditeurImpl.java
com\moi\test\jaxb\impl\LivreImpl.java
com\moi\test\jaxb\impl\LivreTypeImpl.java
```

```
com\moi\test\jaxb\impl\TitreImpl.java
```

Un objet de type factory et des interfaces pour chacun des éléments qui compose le document sont définis.

Pour chaque élément qui peut contenir d'autres éléments, des interfaces de XXXType sont créées (BibliothequeType et LivreType dans l'exemple).

L'interface BibliothequeType définit simplement un getter sur une collection qui contiendra tous les livres.

L'interface LivreType définit des getters et des setters sur les éléments auteur, editeur et titre.

Les interfaces Titre, Editeur et Auteur définissent un getter et un setter sur la valeur des données que ces éléments contiennent.

La classe ObjectFactory permet de créer des instances des différentes entités définies.

Le répertoire impl contient les classes qui implémentent ces interfaces. Ces classes sont spécifiques à l'implémentation des spécifications JAXB utilisées.

Pour pouvoir utiliser ces interfaces et ces classes, il faut les compiler en incluant tous les fichiers .jar contenus dans le répertoire lib de l'implémentation de JAXB au classpath.

33.14.2. L'API JAXB

L'API JAXB propose un framework composé de classes regroupées dans trois packages :

- javax.xml.bind : contient les interfaces principales et la classe JAXBContext
- javax.xml.bind.util : contient des utilitaires
- javax.xml.bind.helper : contient une implémentation partielle de certaines interfaces pour faciliter le développement d'une implémentation des spécifications de JAXB

33.14.3. L'utilisation des classes générées et de l'API

Pour pouvoir utiliser JAXP, il faut tout d'abord obtenir un objet de type JAXBContext qui est le point d'entrée pour utiliser l'API. Il faut utiliser la méthode newInstance() qui attend en paramètre le nom du package qui contient les interfaces générées (celui fournit au paramètre −p de la commande xjc).

Pour pouvoir créer en mémoire les objets qui représentent le document XML, il faut à partir de l'instance du type JAXBContext, appeler la méthode createUnmarshaller() qui renvoie un objet de type Unmarshaller.

L'appel de la méthode unmarshal() permet de créer les différents objets.

Pour parcourir le document, il suffit d'utiliser les différents objets instanciés.

Exemple :

```
package com.moi.test.jaxb;
import javax.xml.bind.*;
import java.io.*;
import java.util.*;
public class TestJAXB {

  public static void main(String[] args) {

    try {
      JAXBContext jc = JAXBContext.newInstance("com.moi.test.jaxb");
      Unmarshaller unmarshaller = jc.createUnmarshaller();
      unmarshaller.setValidating(true);
```

```
      Bibliotheque bibliotheque = (Bibliotheque) unmarshaller.unmarshal(new File("test.xml"));

      List livres = bibliotheque.getLivre();
      for (int i = 0; i < livres.size(); i++) {
        LivreType livre = (LivreType) livres.get(i);
        System.out.println("Livre ");
        System.out.println("Titre   : " + livre.getTitre());
        System.out.println("Auteur  : " + livre.getAuteur());
        System.out.println("Editeur : " + livre.getEditeur());
        System.out.println();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Résultat :

```
Livre
Titre   : titre 1
Auteur  : auteur 1
Editeur : editeur 1
Livre
Titre   : titre 2
Auteur  : auteur 2
Editeur : editeur 2
Livre
Titre   : titre 3
Auteur  : auteur 3
Editeur : editeur 3
```

33.14.4. La création d'un nouveau document XML

Parmi les classes générées à partir du schéma XML, il y a la classe ObjectFactory qui permet de créer des instances des autres classes générées.

Exemple :

```
import javax.xml.bind.*;
import java.io.*;
import java.util.*;
public class TestJAXB2 {

  public static void main(String[] args) {
    try {
      ObjectFactory objFactory = new ObjectFactory();

      Bibliotheque bibliotheque = (Bibliotheque) objFactory.createBibliotheque();
      List livres = bibliotheque.getLivre();
      for (int i = 1; i < 4; i++) {
        LivreType livreType = objFactory.createLivreType();
        //    LivreType livre = objFactory.createLivreType();
        livreType.setAuteur("Auteur" + i);
        livreType.setEditeur("Editeur" + i);
        livreType.setTitre("Titre" + i);
        livres.add(livreType);
      }

    } catch (Exception e) {

    }
  }
}
```

33.14.5. La génération d'un document XML

Une fois la représentation en mémoire du document XML créée ou modifiée, il est fréquent de devoir l'envoyer dans un flux tel qu'un fichier pour conserver les modifications. Cette opération se nomme marshalling.

Pour réaliser cette opération, il faut tout d'abord obtenir un objet du type JAXBContext en utilisant la méthode newInstance(). Cette méthode demande en paramètre une chaîne de caractère indiquant le package des interfaces gérées à partir du schéma.

La méthode createMarshaller() permet d'obtenir un objet de type Marshaller. C'est cet objet qui va formater le document XML.

Il est possible de lui préciser des propriétés pour effectuer sa tache en utilisant la méthode setProperty(). Des constantes sont définies pour ces propriétés dont les principales sont :

- JAXB_ENCODING : permet de préciser le jeux de caractères d'encodage du document XML sous la forme d'une chaîne de caractères
- JAXB_FORMATTED_OUTPUT : booléen qui indique si le document XML doit être formaté

La valeur des propriétés doit être un objet.

L'appel de la méthode marshal() formate le document dont l'objet racine est fourni en premier paramètre. Il existe plusieurs surcharges de cette méthode pour préciser ou est envoyé le résultat de la génération. Le second paramètre permet de préciser cette cible : un flux en sortie, un arbre DOM, des événements SAX.

Exemple :

```
package com.moi.test.jaxb;
import javax.xml.bind.*;
import java.io.*;
import java.util.*;
public class TestJAXB3 {

  public static void main(String[] args) {
    try {

      ObjectFactory objFactory = new ObjectFactory();

      Bibliotheque bibliotheque = (Bibliotheque) objFactory.createBibliotheque();
      List livres = bibliotheque.getLivre();
      for (int i = 1; i < 4; i++) {
        LivreType livreType = objFactory.createLivreType();
        //    LivreType livre = objFactory.createLivreType();
        livreType.setAuteur("Auteur" + i);
        livreType.setEditeur("Editeur" + i);
        livreType.setTitre("Titre" + i);
        livres.add(livreType);
      }
      JAXBContext jaxbContext = JAXBContext.newInstance("com.moi.test.jaxb");
      Marshaller marshaller = jaxbContext.createMarshaller();
      marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, new Boolean(true));
      Validator validator = jaxbContext.createValidator();

      marshaller.marshal(bibliotheque, System.out);
    } catch (Exception e) {
    }
  }
}
```

Résultat :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bibliotheque>
<livre>
<titre>Titre1</titre>
```

```
<auteur>Auteur1</auteur>
<editeur>Editeur1</editeur>
</livre>
<livre>
<titre>Titre2</titre>
<auteur>Auteur2</auteur>
<editeur>Editeur2</editeur>
</livre>
<livre>
<titre>Titre3</titre>
<auteur>Auteur3</auteur>
<editeur>Editeur3</editeur>
</livre>
</bibliotheque>
```

Partie 4 : Développement d'applications d'entreprises

Cette quatrième partie traite d'une utilisation de java en forte expansion : le développement côté serveur. Ce type de développement est poussé par l'utilisation d'internet notamment.

Ces développements sont tellements importants que Sun propose une véritable plate−forme, basée sur le J2SE et orientée entreprise dont les API assurent le développement côté serveur : J2EE (Java 2 Entreprise Edition).

Cette partie regroupe plusieurs chapitres :

- J2EE : introduit la plate−forme Java 2 Entreprise Edition
- Les servlets : plonge au coeur de l'API servlet qui est un des composants de base pour le développement d'applications Web
- Les JSP (Java Servers Pages) : poursuit la discusion avec les servlets en explorant un mécanisme basé sur celles ci pour réaliser facilement des pages web dynamiques
- JSTL (Java server page Standard Tag Library) : est un ensemble de bibliothèques de tags personnalisés communément utilisé dans les JSP
- Les frameworks pour les applications web : facilitent le développement d'applications web en fournissant un socle de construction et une structuration des développements
- Java Server Faces : détaille l'utilisation de la technologie Java Server Faces (JSF) dont le but est de proposer un framework qui facilite et standardise le développement d'applications web avec Java.
- JNDI (Java Naming and Directory Interface) : introduit l'API capable d'acceder aux services de nommage et d'annuaires
- JMS (Java Messaging Service) : indique comment utiliser cette API qui permet l'utilisation de système de messages pour l'échange de données entre applications
- JavaMail : traite de l'API qui permet l'envoie et la réception d'e mail

- Les EJB (Entreprise Java Bean) : propose une présentation del'API et les spécifications pour des objets chargés de contenir les règles métiers
- Les services web : permettent l'appels de services distants en utilisant un protocole de communication et une structuration des données échangées avec XML de façon standardisée

34. J2EE

J2EE est l'acronyme de Java 2 Entreprise Edition. Cette édition est dédiée à la réalisation d'applications pour entreprises. J2EE est basé sur J2SE (Java 2 Standard Edition) qui contient les API de base de java.

Ce chapitre contient plusieurs sections :

- Présentation de J2EE : présente rapidement la plate−forme J2EE
- Les API de J2EE : présente rapidement les différentes API qui composent J2EE
- L'environnement d'exécution des applications J2EE : présente les différents élements qui compose l'environnement d'exécution des applications J2EE
- L'assemblage et le déploiement d'applications J2EE : décrit le mode d'assemblage et de déploiement des applications J2EE
- J2EE 1.4 SDK : installation et prise en main du J2EE 1.4 SDK

34.1. Présentation de J2EE

J2EE est une plate−forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :

- un ensemble de spécifications pour une infrastructure dans laquelle s'exécute les composants écrits en java : un tel environnement se nomme serveur d'application.
- un ensemble d'API qui peuvent être obtenues et utilisées séparement. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.

Sun propose une implémentation minimale des spécifications de J2EE : le J2EE SDK. Cette implémentation permet de développer des applications respectant les spécifications mais n'est pas prévue pour être utilisée dans un environnement de production. Ces spécifications doivent être respectées par les outils développés par des éditeurs tiers.

L'utilisation de J2EE pour développer et exécuter une application propose plusieurs avantages :

- une architecture d'application basée sur les composants qui permet un découpage de l'application et donc une séparation des rôles lors du développement
- la possibilité de s'interfacer avec le système d'information existant grace à de nombreuses API : JDBC, JNDI, JMS, JCA ...
- la possibilité de choisir les outils de développement et le ou les serveurs d'applications utilisés qu'ils soient commerciaux ou libres

J2EE permet une grande fléxibilité dans le choix de l'architecture de l'application en combinant les differents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de J2EE. L'architecture d'une application se découpe idéalement en au moins trois tiers :

- la partie cliente : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composée d'une application standalone, d'une application web ou d'applets
- la partie métier : c'est la partie qui encapsule les traitements (dans des EJB ou des JavaBeans)
- la partie données : c'est la partie qui stocke les données

La suite de ce chapitre sera développée dans une version future de ce document

34.2. Les API de J2EE

J2EE regroupe un ensemble d'API pour le développement d'applications d'entreprise.

API	Rôle	version de l'API dans		
		J2EE 1.2	J2EE 1.3	J2EE 1.4
Entreprise java bean (EJB)	Composants serveurs contenant la logique métier	1.1	2.0	2.1
Remote Méthod Invocation (RMI) et RMI−IIOP	RMI permet d'utilisation d'objet java distribué. RMI−IIOP est une extension de RMI pour utiliser avec CORBA.	1.0		
Java Naming and Directory Interface (JNDI)	Accès aux services de nommage et aux annuaires d'entreprises	1.2	1.2	1.2.1
Java Database Connectivity (JDBC)	Accès aux bases de données. J2EE intègre une extension de cette API	2.0	2.0	3.0
Java Transaction API (JTA) Java Transaction Service (JTS)	Support des transactions	1.0	1.0	1.0
Java Messaging service (JMS)	Support de messages via des MOM (Messages Oriented Middleware)	1.0	1.0	1.1
Servlets	Composants basés sur le concept C/S pour ajouter des fonctionnalités à un serveur. Pour le moment, principalement utilisé pour étendre un serveur web	2.2	2.3	2.4
Java Server Pages (JSP)		1.1	1.2	2.0
Java IDL	Utilisation de CORBA			
JavaMail	Envoie et réception d'email	1.1	1.2	1.3
J2EE Connector Architecture (JCA)	Connecteurs pour accéder à des ressources du système d'information de l'entreprises tel que CICS, TUXEDO, SAP ...		1.0	1.5
Java API for XML Parsing (JAXP)	Analyse et exploitation de données au format XML		1.1	1.2
Java Authentication and Authorization Service (JAAS)	Echange sécurisé de données		1.0	
JavaBeans Activation Framework	Utilisé par JavaMail : permet de déterminer le type mime		1.0.2	1.0.2
Java API for XML−based RPC (JAXP−RPC)				1.1
SOAP with Attachments API for Java (SAAJ)				1.2
Java API for XML Registries (JAXR)				1.0

Java Management Extensions (JMX)				1.2
Java Authorization Service Provider Contract for Containers (JACC)				1.0

Ces API peuvent être regroupées en trois grandes catégories :

- les composants : Servlet, JSP, EJB
- les services : JDBC, JTA/JTS, JNDI, JCA, JAAS
- la communication : RMI−IIOP, JMS, Java Mail

34.3. L'environnement d'exécution des applications J2EE

J2EE propose des spécifications pour une infrastructure dans laquelle s'exécute les composants. Ces spécifications décrivent les rôles de chaque éléments et précisent un ensemble d'interfaces pour permettre à chacun de ces éléments de communiquer.

Ceci permet de séparer les applications et l'environnement dans lequel il s'exécute. Les spécifications précisent à l'aide des API un certain nombre de fonctionnalités que doivent implémenter l'environnement d'exécution. Ces fonctionnalités sont de bas niveau ce qui permet aux développeurs de se concentrer sur la logique métier.

Pour éxecuter ces composants de natures différentes, J2EE défini des conteneurs pour chacun de ces composants. Il définit pour chaque composants des interfaces qui leur permettront de dialoguer avec les composants lors de leur execution. Les conteneurs permettent aux applications d'accéder aux ressources et aux services en utilisant les API.

Les appels aux composants se font par des clients via les conteneurs. Les clients n'accèdent pas directement aux composants mais sollicite le conteneur pour les utiliser.

34.3.1. Les conteneurs

Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Les conteneurs fournissent des services qui peuvent être utilisés par les applications lors de leur exécution.

Il existe plusieurs conteneurs définit par J2EE:

- conteneur web : pour exécuter les servlets et les JSP
- conteneur d'EJB : pour exécuter les EJB
- conteneur client : pour executer des applications standalone sur les postes qui utilisent des composants J2EE

Les serveurs d'applications peuvent fournir un conteneur web uniquement (exemple : Tomcat) ou un conteneur d'EJB uniquement (exemple : JBoss, Jonas, ...) ou les deux (exemple : Websphere, Weblogic, ...).

Pour déployer une application dans un conteneur, il faut lui fournir deux éléments :

- l'application avec tous les composants (classes compilées, ressources ...) regroupée dans une archive ou module. Chaque conteneur possède son propre format d'archive.
- un fichier descripteur de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application

Il existe trois types d'archives :

Archive / module	Contenu	Extension	Descripteur de déploiement

bibliothèque	Regroupe des classes	jar	
application client	Regroupe les ressources necessaires à leur execution (classes, bibliothèques, images, ...)	jar	application−client.jar
web	Regroupe les servlets et les JSP ainsi que les ressources necessaires à leur execution (classes, bibliothèques de balises, images, ...)	war	web.xml
EJB	Regroupe les EJB et leur composants (classes)	jar	ejb−jar.xml

Une application est un regroupement d'un ou plusieurs modules dans un fichier EAR (Entreprise ARchive). L'application est décrite dans un fichier application.xml lui même contenu dans le fichier EAR

34.3.2. Le conteneur web

Le conteneur web est une implémentation des spécifications servlets et par extension des spécifications des JSP. Ce type de conteneur est composé de deux éléments majeur : un moteur de servlets (servlets engine) et un moteur de JSP (JSP engine).

Les conteneurs web peuvent généralement utiliser leur propre serveur web et être utilisés en tant que plug in d'un serveur web dédié (Apache, IIS, ...).

L'implémentation de référence pour ce type de conteneur est le projet open source Tomcat du groupe Apache.

Les servlets et les JSP détaillées dans des chapitres dédiés.

34.3.3. Le conteneur d'EJB

Les EJB sont détaillées dans un chapitre dédié.

34.3.4. Les services proposés par la plate−forme J2EE

Une plate−forme d'execution J2EE complète implementée dans un serveur d'application propose les services suivants :

- service de nommage (naming service)
- service de déploiement (deployment service)
- service de gestion des transactions (transaction service)
- service de sécurité (security service)

Ces services sont utilisés directement ou indirectement par les conteneurs mais aussi par les composants qui s'exécutent dans les conteneurs grace à leurs API respectives.

34.4. L'assemblage et le déploiement d'applications J2EE

J2EE propose une spécification pour décrire le mode d'assemblage et de déploiement d'une application J2EE.

Une application J2EE peut regrouper différents modules : modules web, modules EJB ... Chacun de ces modules possède son propre mode de packaging. J2EE propose de regrouper ces différents modules dans un module unique sous la forme d'un fichier EAR (Entreprise ARchive).

Le format de cette archive est très semblable à celui des autres archives :

- un contenu : les différents modules qui composent l'application (module web, EJB, fichier RAR ...)
- un fichier descripteur de déploiement

Les serveurs d'application extraient chaque module du fichier EAR et les déploient séparément un par un.

34.4.1. Le contenu et l'organisation d'un fichier EAR

Le fichier EAR est composé au minimum :

- d'un ou plusieurs modules
- un répertoire META−INF contenant un fichier descripteur de déploiement nommé application.xml

Les modules ne doivent pas obligatoirement être insérés à la racine du fichier EAR : ils peuvent être mis dans un des sous répertoires pour organiser le contenu de l'application. Il est par exemple pratique de créer un répertoire lib qui contient les fichier .jar des bibliothèques communes aux différents modules.

34.4.2. La création d'un fichier EAR

Pour créer un fichier EAR, il est possible d'utiliser un outil graphique fourni par le vendeur du serveur d'application ou de créer le fichier manuellement en suivant les étapes suivantes :

1. créer l'arboresence des répertoires qui vont contenir les modules
2. insérer dans cette arborescence les différents modules à inclure dans le fichier EAR
3. créer le répertoire META−INF (en respectant la casse)
4. créer le fichier application.xml dans ce répertoire
5. utiliser l'outil jar pour créer le fichier le fichier EAR en précisant les options cvf, le nom du fichier ear avec son extension et les différents éléments qui compose le fichier (modules, répertoire dont le répertoire META−INF).

34.4.3. Les limitations des fichiers EAR

Actuellement les fichiers EAR ne servent qu'à regrouper différents modules pour former une seule entité. Rien n'est actuellement prévu pour prendre en compte la configuration des objets permettant l'accès aux ressources par l'application tel qu'une base de données (JDBC pour DataSource, pool de connexion ...), un système de message (JMS), etc ...

Pour lever une partie de ces limites, les serveurs d'applications commerciaux proposent souvent des mécanismes propriétaires supplémentaires pour palier à ces manques en attendant une évolution des spécifications.

34.5. J2EE 1.4 SDK

La version 1.4 de J2EE a été diffusée en novembre 2003.

La grande nouveauté de la version 1.4 est le support des services web. Deux nouvelles API ont été ajoutées pour normaliser le déploiement (J2EE deployment API 1.1) et la gestion des applications(J2EE management API 1.0 qui

utilise JMX). Une nouvelle API permet de standardiser l'authentification (Java ACC : Java Authorization Contract for Container). Plusieurs API déjà présentes dans les précédentes versions de J2EE ont été mises à jour (EJB, JSP, Servlet, ...) :

- J2EE Connector Architecture 1.5
- Enterprise JavaBeans (EJB) 2.1
- JavaServer Pages (JSP) 2.0
- Java Servlet 2.4
- JavaMail 1.3
- Java Message Service 1.1
- Java API for XML parsing (JAXP) 1.2
- Java API for XML−based RPC (JAX−RPC) 1.1
- SOAP with Attachments API for Java (SAAJ) 1.2
- Java API for XML Registries (JAXR) 1.0
- Java Management Extensions (JMX) 1.2
- Java Authorization Service Provider Contract for Containers (JACC) 1.0

Le J2EE SDK 1.4 qui est l'implémentation de référence inclus le J2SE SDK 1.4.2 et J2EE 1.4 application server.

34.5.1. Installation de l'implémentation de référence sous Windows

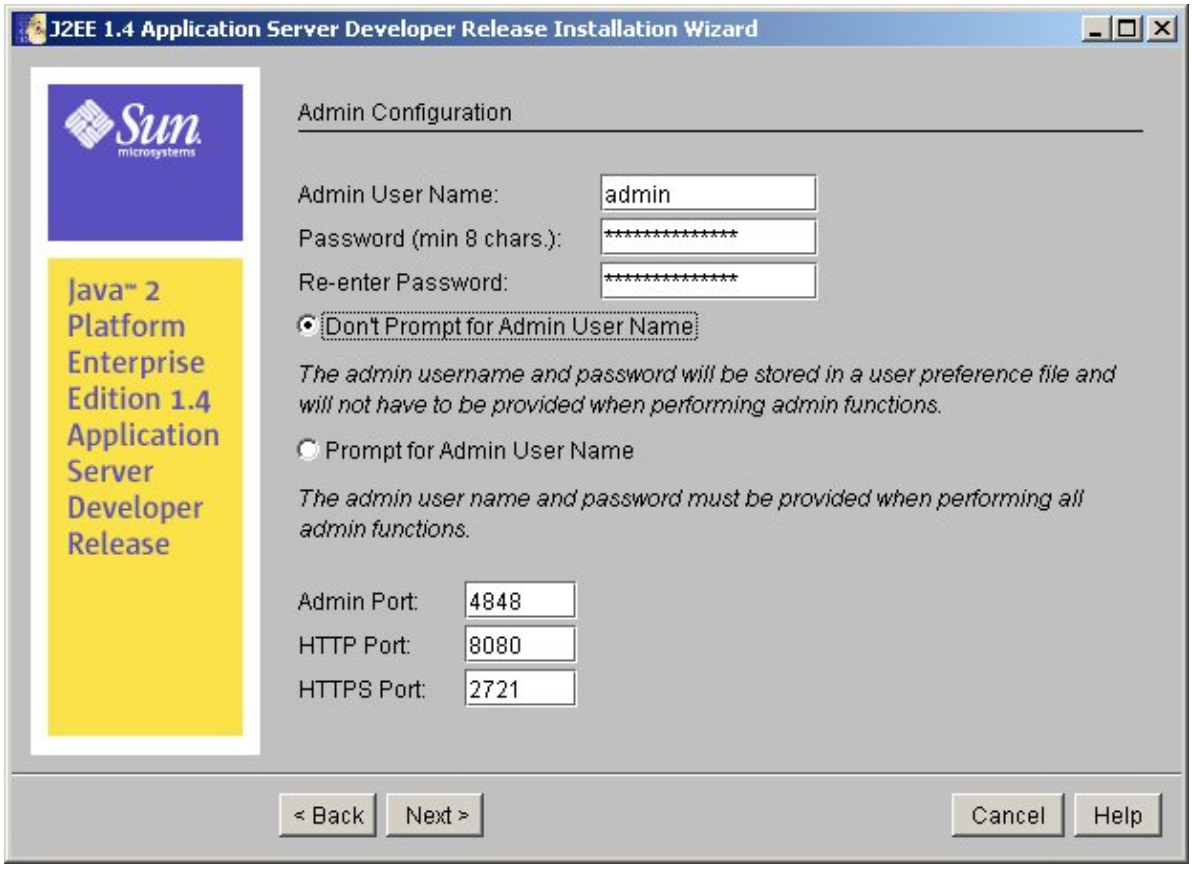
Le J2EE SDK 1.4 peut être installé sur les systèmes Microsoft suivants : Windows 2000 pro avec un service pack SP2, Windows XP PRO avec un service pack SP1 et Windows Server 2003.

Il existe plusieurs packages d'installation : celui utilisé ci dessous ne contient que le serveur d'application puisque le J2SE 1.4.2 était déjà présent sur la machine.

Lancer le programme j2eesdk−1_4−dr−windows−eval−app.exe. L'application extrait les fichiers, lance le J2RE et exécute un assistant qui va guider l'installation :

- la première page est la page d'accueil (welcome) : cliquez sur le bouton « Next »
- La page suivant permet de lire et d'accepter la licence d'utilisation (software licence agreement) : lire la licence et si vous l'acceptez, cliquez sur le bouton « Yes » puis sur le bouton « Next »
- la page suivante permet de sélectionner le répertoire d'installation du produit (Select Installation Directory) : sélectionnez le répertoire d'installation et cliquez sur « Next »

  Cliquez sur « Create directory »
- la page suivante permet de préciser l'emplacement du J2SDK nécessaire au produit (Java 2 SDK Required) : sélectionnez l'emplacement du J2SE SDK (si il est présent sur la machine, son chemin est proposé par défaut), puis cliquez sur le bouton « Next ».
- la page suivante permet de préciser les informations nécessaires la configuration du serveur



Il faut saisir des paramètres de configuration : saisir le mot de passe de l'administrateur et sélectionner l'option pour l'authentification ou non lors d'actions d'administration

Il est aussi possible de définir les ports pour le module d'administration et pour les serveurs web HTTP et HTTPS.

Une fois les informations saisies, cliquez sur le bouton « Next ».

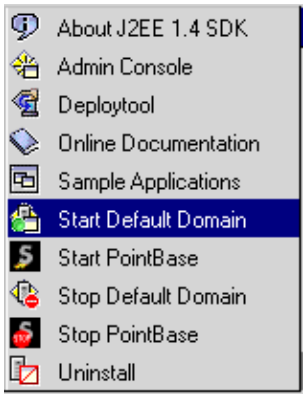
- La page suivante permet de sélectionner le type d'installation à réaliser (Installation Options) : pour une première installation, il suffit de conserver l'option par défaut proposée puis cliquez sur le bouton « Next »
- La page suivante synthétise les différentes informations avant l'installation (Ready to Install) : cliquez sur « Install Now »
- La dernière page indique que l'installation s'est bien terminée et donne quelques informations sommaires sur les différents outils

Il est utile de rajouter le répertoire bin du répertoire d'installation de J2EE SDK 1.4 à la variable PATH du système d'exploitation.

34.5.2. Démarrage et arrêt du serveur

Un domaine permet de regrouper des applications avec une configuration particulière qui s'exécutent sur une instance particulière du serveur. Lors de l'installation un domaine par défaut est créé : domain1

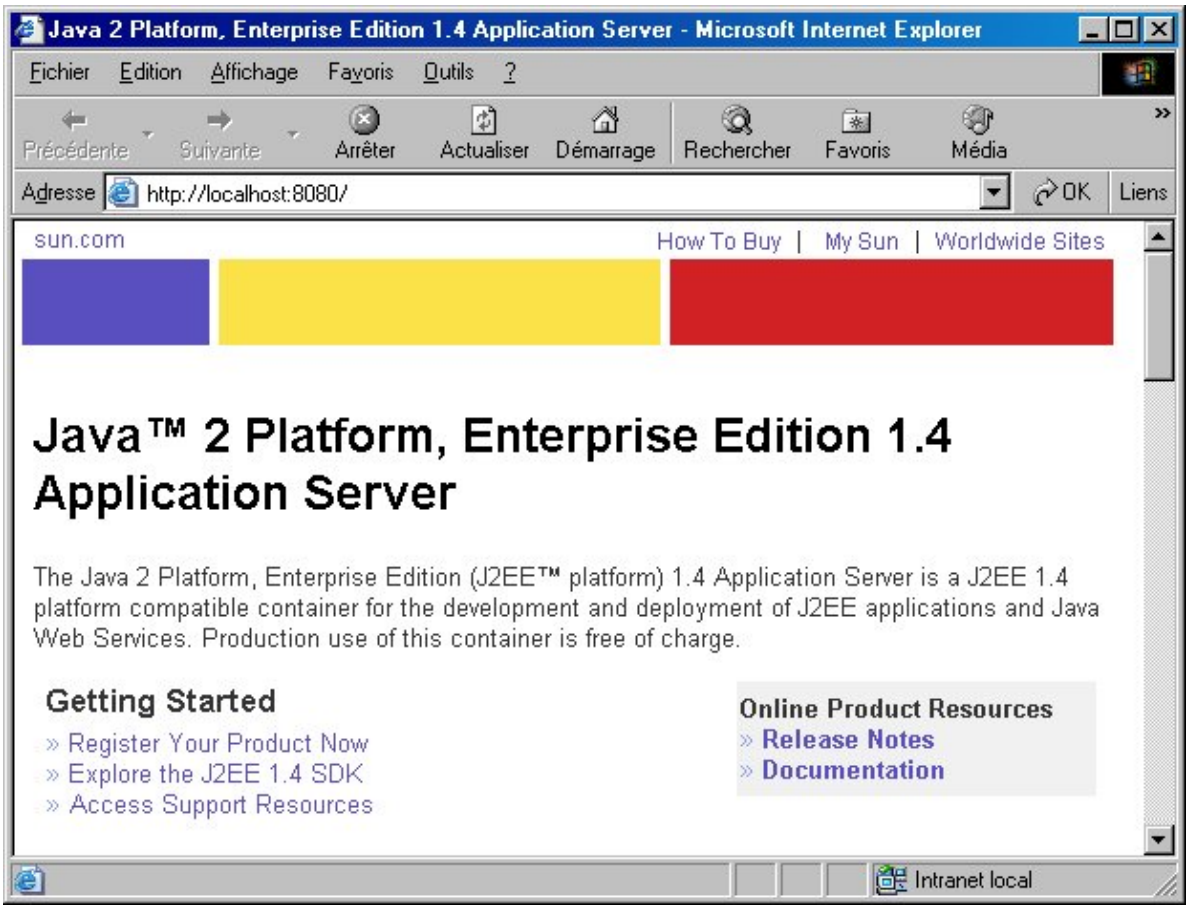
Le programme d'installation a créé plusieurs entrées dans le menu « Démarrer/Programmes/Sun Microsystems/J2EE 1.4 SDK/ »



L'option « Start default domain » permet de démarrer le domaine domain1.

Il suffit d'appuyer sur une touche pour fermer la fenêtre.

Pour vérifier la bonne exécution du serveur, il suffit d'appeler l'URL http://localhost:nnn/ dans un navigateur ou nnnn représente le port http précisé dans les paramètres lors de l'installation.



L'arrêt du domaine par défaut peut être obtenu en utilisant l'option « Stop default domain ».

34.5.3. L'outil asadmin

J2EE application server est livré avec une application nommée asadmin, utilisable sur une ligne de commandes, pour administrer le serveur.

Cette application utilise deux modes de fonctionnement :

- la réception de commandes par la console après son lancement
- la passage des commandes en argument de la commande

Les commandes possèdent des noms bien définis en fonction de leurs actions et nécessite souvent un ou plusieurs paramètres.

Exemple : démarrage d'un domaine

C:\>asadmin start−domain domain1

Starting Domain domain1, please wait.

Log redirected to C:\Sun\AppServer\domains\domain1\logs\server.log.

Domain domain1 started.

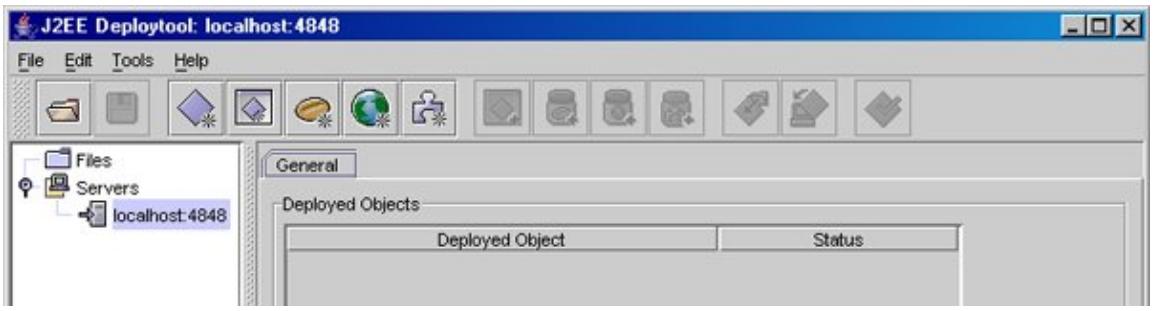
Exemple : arrêt d'un domaine

C:\>asadmin stop−domain domain1

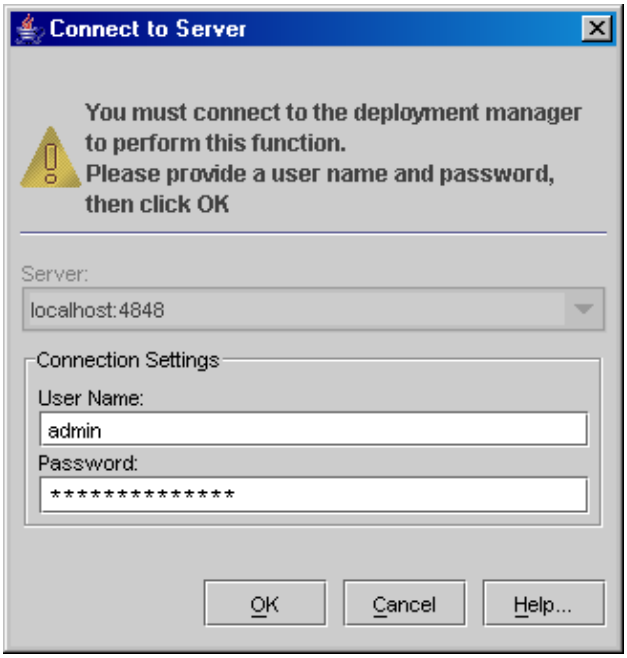
Domain domain1 stopped.

34.5.4. Le déploiement d'application

Pour déployer une application sous la forme d'un fichier war ou ear il suffit de copier le fichier dans le sous répertoire domains/nom_du_domaine/autodeploy.



Il est nécessaire de s'authentifier auprès du serveur d'application pour certaines opérations.



34.5.5. La console d'administration

La console d'administration est une application web qui permet de configurer le serveur.

Pour l'utiliser, le serveur doit être lancé et il suffit de saisir dans un navigateur l'Url http://localhost:4848/asadmin

35. Les servlets

Les serveurs web sont de base uniquement capables de renvoyer des fichiers présents sur le serveur en réponse à une requête d'un client. Cependant, pour permettre l'envoi d'une page HTML contenant par exemple une liste d'articles répondant à différents critères, il faut créer dynamiquement ces pages HTML. Plusieurs solutions existent pour ces traitements. Les servlets java sont une des ces solutions.

Mais les servlets peuvent aussi servir à d'autres usages.

Sun fourni des informations sur les servlets sur son site : http://java.sun.com/products/servlet/index.html

Ce chapitre contient plusieurs sections :

35.1. Présentation des servlets

Une servlet est un programme qui s'exécute côté serveur en tant qu'extension du serveur. Elle reçoit une requête du client, elle effectue des traitements et renvoie le résultat. La liaison entre la servlet et le client peut être directe ou passer par un intermédiaire comme par exemple un serveur http.

Même si pour le moment la principale utilisation des servlets est la génération de pages html dynamiques utilisant le protocole http et donc un serveur web, n'importe quel protocole reposant sur le principe de requête/réponse peut faire usage d'une servlet.

Ecrite en java, une servlet en retire ses avantages : la portabilité, l'accès à toutes les API de java dont JDBC pour l'accès aux bases de données, ...

Une servlet peut être invoquée plusieurs fois en même temps pour répondre à plusieurs requêtes simultanées.

La servlet se positionne dans une architecture Client/Serveur trois tiers dans le tiers du milieu entre le client léger chargé de l'affichage et la source de données.

Il existe plusieurs versions des spécifications de l'API Servlets :

Version	
2.0	1997

2.1	Novembre 1998, partage d'informations grace au Servletcontext La classe GenericServlet implémente l'interface ServletConfig une méthode log() standard pour envoyer des informations dans le journal du conteneur objet RequestDispatcher pour le transfert du traitement de la requête vers une autre ressource ou inclure le résultat d'une autre ressource
2.2	Aout 1999, format war pour un déploiement standard des applications web mise en buffer de la réponse inclus dans J2EE 1.2
2.3	Septembre 2001, JSR 053 : nécessite le JDK 1.2 minimum ajout d'un mécanisme de filtre ajout de méthodes pour la gestion d'évenements liés à la création et la destruction du context et de la session inclus dans J2EE 1.3
2.4	Novembre 2003, JSR 154 inclus dans J2EE 1.4

35.1.1. Le fonctionnement d'une servlet (cas d'utilisation de http)

Un serveur d'application permet de charger et d'exécuter les servlets dans une JVM. C'est une extension du serveur web. Ce serveur d'application contient entre autre un moteur de servlets qui se charge de manager les servlets qu'il contient.

Pour exécuter une servlet, il suffit de saisir une URL qui désigne la servlet dans un navigateur.

1. Le serveur reçoit la requête http qui necessite une servlet de la part du navigateur
2. Si c'est la premiere sollicitation de la servlet, le serveur l'instancie. Les servlets sont stockées (sous forme de fichier .class) dans un répertoire particulier du serveur. Ce répertoire dépend du serveur d'application utilisé. La servlet reste en mémoire jusqu'à l'arret du serveur. Certains serveurs d'application permettent aussi d'instancier des servlets dès le lancement du serveur.

   La servlet en mémoire, peut être appelée par plusieurs threads lancés par le serveur pour chaque requête. Ce principe de fonctionnement évite d'instancier un objet de type servlet à chaque requête et permet de maintenir un ensemble de ressources actives tel qu'une connection à une base de données.
3. le serveur créé un objet qui représente la requête http et objet qui contiendra la réponse et les envoie à la servlet
4. la servlet créé dynamiquement la reponse sous forme de page html transmise via un flux dans l'objet contenant la réponse. La création de cette réponse utilise bien sûre la requête du client mais aussi un ensemble de ressources incluses sur le serveur tels de que des fichiers ou des bases de données.
5. le serveur récupère l'objet réponse et envoie la page html au client.

35.1.2. Les outils nécessaires pour développer des servlets

Initialement, pour développer des servlets avec le JDK standard édition, il faut utiliser le Java Server Development Kit (JSDK) qui est une extension du JDK. Pour réaliser les tests, le JSDK fournit, dans sa version 2.0 un outil nommé servletrunner et depuis sa version 2.1, il fournit un serveur http allégé.

Actuellement, pour exécuter des applications web, il faut utiliser un conteneur web ou serveur d'application : il existe de nombreuses versions commerciales tel que IBM Webpshere ou BEA WebLogic mais aussi des versions libres tel que Tomcat du projet GNU Jakarta.

Ce serveur d'application ou ce conteneur web doit utiliser ou inclure un serveur http dont le plus utilisé est Apache.

Le choix d'un serveur d'application ou d'un conteneur web doit tenir compte de la version du JSDK qu'il supporte pour être compatible avec celle utilisée pour le développement des servlets. Le choix entre un serveur commercial et un libre doit tenir compte principalement du support technique, des produits annexes fournis et des outils d'installation et de configuration.

Pour simplement développer des servlets, le choix d'un serveur libre se justifie pleinenement de part leur gratuité et leur « légèreté ».

35.1.3. Le role du conteneur web

Un conteneur web est un moteur de servlet qui prend en charge et gère les servlets : chargement de la servlet, gestion de son cycle de vie, passage des requêtes et des réponses ... Un conteur web peut être intégré dans un serveur d'application qui va contenir d'autre conteneur et eventuellement proposer d'autres services..

Le chargement et l'instanciation d'une servlet se font selon le paramétrage soit au lancement du serveur soit à la première invocation de la servlet. Dès l'instanciation, la servlet est initialisée une seule et unique fois avant de pouvoir répondre aux requêtes. Cette initialisation peut permettre de mettre en place l'accès à des ressources tel qu'une base de données.

35.1.4. Les différences entre les servlets et les CGI

Les programmes ou script CGI (Common Gateway Interface) sont aussi utilisés pour générer des pages HTML dynamiques. Ils représentent la plus ancienne solution pour réaliser cette tache.

Un CGI peut être écrit dans de nombreux langages.

Il existe plusieurs avantages à utiliser des servlets plutôt que des CGI :

- la portabilité offerte par Java bien que certains langages de script tel que PERL tourne sur plusieurs plate−formes.
- la servlet reste en mémoire une fois instanciée ce qui permet de garder des ressources systèmes et gagner le temps de l'initialisation. Un CGI est chargé en mémoire à chaque requête, ce qui réduit les performances.
- les servlets possèdent les avantages de toutes les classes écrites en java : accès aux API, aux java beans, le garbage collector, ...

35.2. L'API servlet

Les servlets sont conçues pour agir selon un modele de requête/reponse. Tous les protocoles utilisant ce modèle peuvent être utilisés tel que http, ftp, etc ...

L'API servlets est une extension du jdk de base, et en tant que telle elle est regroupée dans des packages préfixés par javax

L'API servlet regroupe un ensemble de classes dans deux packages :

- javax.servlet : contient les classes pour développer des serlvets génériques indépendantes d'un protocole
- javax.servlet.http : contient les classes pour développer des servlets qui reposent sur le protocole http utilisé par les serveurs web.

Le package javax.servlet définit plusieurs interfaces, méthodes et exceptions :

javax.servlet	Nom	Role
Les interfaces	RequestDispatcher	Définition d'un objet qui permet le renvoi d'une requête vers une autre ressource du serveur (une autre servlet, une JSP ...)
	Servlet	Définition de base d'une servlet
	ServletConfig	Définition d'un objet pour configurer la servlet
	ServletContext	Définition d'un objet pour obtenir des informations sur le contexte d'execution de la servlet

	ServletRequest	Définition d'un objet contenant la requête du client
	ServletResponse	Définition d'un objet qui contient la reponse renvoyée par la servlet
	SingleThreadModel	Permet de définir une servlet qui ne répondra qu'à une seule requête à la fois
Les classes	GenericServlet	Classe définissant une servlet indépendante de tout protocole
	ServletInputStream	Flux permet la lecture des donnes de la requête cliente
	ServletOutPutStream	Flux permettant l'envoie de la reponse de la servlet
Les exceptions	SevletException	Exception générale en cas de problème durant l'exécution de la servlet
	UnavailableException	Exception levée si la servlet n'est pas disponible

Le package javax.servlet.http définit plusieurs interfaces et méthodes :

Javax.servlet	Nom	Role
Les interfaces	HttpServletRequest	Hérite de ServletRequest : définit un objet contenant une requête selon le protocole http
	HttpServletResponse	Hérite de ServletResponse : définit un objet contenant la reponse de la servlet selon le protocole http
	HttpSession	Définit un objet qui représente une session
Les classes	Cookie	Classe représentant un cookie (ensemble de données sauvegardées par le brower sur le poste client)
	HttpServlet	Hérite de GenericServlet : classe définissant une servlet utilisant le protocole http
	HttpUtils	Classe proposant des méthodes statiques utiles pour le développement de servlet http

35.2.1. L'interface Servlet

Une servlet est une classe Java qui implémente l'interface javax.servlet.Servlet. Cette interface définit 5 méthodes qui permettent au conteneur web de dialoguer avec la servlet : elle encapsule ainsi les méthodes nécessaires à la communication entre le conteneur et la servlet.

Méthode	Rôle
void service (ServletRequest req, ServletResponse res)	Cette méthode est exécutée par le conteneur lorsque la servlet est sollicitée : chaque requête du client déclenche une seule exécution de cette méthode. Cette méthode pouvant être exécutée par plusieurs threads, il faut prévoir un processus d'exclusion pour l'utilisation de certaines ressources.
void init(ServletConfig conf)	Initialisation de la servlet. Cette méthode est appelée une seule fois après l'instanciation de la servlet. Aucun traitement ne peut être effectué par la servlet tant que l'exécution de cette méthode n'est pas terminée.
ServletConfig getServletConfig()	Renvoie l'objet ServletConfig passé à la méthode init
void destroy()	Cette méthode est appelée lors de la destruction de la servlet. Elle permet de liberer proprement certaines ressources (fichiers, bases de données ...). C'est le serveur qui appelle cette méthode.

String getServletInfo()	Renvoie des informations sur la servlet.

Les méthodes init(), service() et destroy() assurent le cycle de vie de la servlet en étant respectivement appellées lors de la création de la servlet, lors de son appel pour le traitement d'une requête et lors de sa destruction.

La méthode init() est appellée par le serveur juste après l'instanciation de la servlet.

La méthode service() ne peut pas être invoquée tant que la méthode init() n'est pas terminée.

La méthode destroy() est appelée juste avant que le serveur ne détruise la servlet : cela permet de liberer des ressources allouées dans la méthode init() tel qu'un fichier ou une connexion à une base de données.

35.2.2. La requête et la réponse

L'interface ServletRequest définit plusieurs méthodes qui permettent d'obtenir des données sur la requête du client :

Méthode	Role
ServletInputStream getInputStream()	Permet d'obtenir un flux pour les données de la requête
BufferedReader getReader()	Idem

L'interface ServletResponse définit plusieurs méthodes qui permettent de fournir la réponse faite par la servlet suite à ces traitements :

Méthode	Role
SetContentType	Permet de préciser le type MIME de la réponse
ServletOutputStream getOutputStream()	Permet d'obtenir un flux pour envoyer la réponse
PrintWriter getWriter()	Permet d'obtenir un flux pour envoyer la réponse

35.2.3. Un exemple de servlet

Une servlet qui implémente simplement l'interface Servlet doit évidemment redéfinir toutes les méthodes définies dans l'interface.

Il est très utile lorsque que l'on crée une servlet qui implémente directement l'interface Servlet de sauvegarder l'objet ServletConfig fourni par le conteneur en paramètre de la méthode init() car c'est le seul moment où l'on a accès à cet objet.

Exemple (code Java 1.1) :

```
import java.io.*;
import javax.servlet.*;

public class TestServlet implements Servlet {
  private ServletConfig cfg;

  public void init(ServletConfig config) throws ServletException {
     cfg = config;
  }

  public ServletConfig getServletConfig() {
    return cfg;
  }

  public String getServletInfo() {
```

```
    return "Une servlet de test";
  }

  public void destroy() {
  }

  public void service (ServletRequest req,  ServletResponse res )
  throws ServletException, IOException  {
    res.setContentType( "text/html" );
    PrintWriter out = res.getWriter();
    out.println( "<THML>" );
    out.println( "<HEAD>");
    out.println( "<TITLE>Page generee par une servlet</TITLE>" );
    out.println( "</HEAD>" );
    out.println( "<BODY>" );
    out.println( "<H1>Bonjour</H1>" );
    out.println( "</BODY>" );
    out.println( "</HTML>" );
    out.close();
  }
}
```

35.3. Le protocole HTTP

Le protocole HTTP est un protocole qui fonctionne sur le modèle client/serveur. Un client qui est une application (souvent un navigateur web) envoie une requête à un serveur (un serveur web). Ce serveur attend en permanence les requêtes sur un port particulier (par défaut le port 80). A la réception de la requête, le serveur lance un thread qui va la traiter pour générer la réponse. Le serveur renvoie la réponse au client une fois les traitements terminés.

Une particularité du protocole HTTP est de maintenir la connexion entre le client et le serveur uniquement durant l'échange de la requête et de la réponse.

Il existe deux versions principales du protocole HTTP : 1.0 et 1.1.

La requête est composée de trois parties :

- la commande
- la section en−tête
- le corps

La première ligne de la requête contient la commande à exécuter par le serveur. La commande est suivie éventuellement d'un argument qui précise la commande (par exemple l'url de la ressource demandée). Enfin la ligne doit contenir la version du protocole HTTP utilisé, précédée de HTTP/.

Exemple :

GET / index.html HTTP/1.0

Avec HTTP 1.1, les commandes suivantes sont définies : GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE et CONNECT. Les trois premières sont les plus utilisées.

Il est possible de fournir sur les lignes suivantes de la partie en−tête des paramètres supplémentaires. Cette partie en−tête est optionnelle. Les informations fournies peuvent permettre au serveur d'obtenir des informations sur le client. Chaque information doit être mise sur une ligne unique. Le format est nom_du_champ:valeur. Les champs sont prédéfinis et sont sensibles à la casse.

Une ligne vide doit précéder le corps de la requête. Le contenu du corps de la requête dépend du type de la commande.

La requête doit obligatoirement être terminée par une ligne vide.

La réponse est elle aussi composée des trois mêmes parties :

- une ligne de status

- un en−tête dont le contenu est normalisé
- un corps dont le contenu dépend totalement de la requête

La première ligne de l'en−tête contient un état qui est composé : de la version du protocole HTTP utilisé, du code de status et d'une description succinte de ce code.

Le code de status est composé de trois chiffres qui donnent des informations sur le résultat du traitement qui a généré cette réponse. Ce code peut être regroupé en plusieurs catégories en fonction de leur valeur :

Plage de valeur du code	Signification
100 à 199	Information
200 à 299	Traitement avec succès
300 à 399	La requête a été redirigée
400 à 499	La requête est incomplète ou erronée
500 à 599	Une erreur est intervenue sur le serveur

Plusieurs codes sont définis par le protocole HTTP dont les plus importants sont :

- 200 : traitement correct de la requête
- 204 : traitement correct de la requête mais la réponse ne contient aucun contenu (ceci permet au browser de laisser la page courante affichée)
- 404 : la ressource demande n'est pas trouvée (sûrement le plus célèbre)
- 500 : erreur interne du serveur

L'en−tête contient des informations qui précise le contenu de la réponse.

Le corps de la réponse est précédé par une ligne vide.

La suite de cette section sera développée dans une version future de ce document

35.4. Les servlets http

L'usage principal des servlets est la création de pages HTML dynamiques. Sun fourni une classe qui encapsule une servlet utilisant le protocole http. Cette classe est la classe HttpServlet.

Cette classe hérite de GenericServlet, donc elle implémente l'interface Servlet, et redéfinie toutes les méthodes nécessaires pour fournir un niveau d'abstraction permettant de développer facilement des servlets avec le protocole http.

Ce type de servlet n'est pas utile seulement pour générer des pages HTML bien que cela soit son principale usage, elle peut aussi réaliser un ensemble de traitements tel que mettre à jour une base de données. En réponse, elle peut générer une page html qui indique le succès ou non de la mise à jour. Une servlets peut aussi par exemple renvoyer une image qu'elle aura dynamiquement générée en fonction de certains paramètres.

Elle définit un ensemble de fonctionnalités très utiles : par exemple, elle contient une méthode service() qui appelle certaines méthodes à redéfinir en fonction du type de requête http (doGet(), doPost(), etc ...).

La requête du client est encapsulée dans un objet qui implémente l'interface HttpServletRequest : cet objet contient les données de la requête et des informations sur le client.

La réponse de la servlet est encapsulée dans un objet qui implémente l'interface HttpServletResponse.

Typiquement pour définir une servlet, il faut définir une classe qui hérite de la classe HttpServlet et redéfinir la classe doGet et/ou doPost selon les besoins.

La méthode service héritée de HttpServlet appelle l'une ou l'autre de ces méthodes en fonction du type de la requête http :

- une requête GET : c'est une requête qui permet au client de demander une ressource
- une requête POST : c'est une requête qui permet au client d'envoyer des informations issues par exemple d'un formulaire

Une servlet peut traiter un ou plusieurs types de requêtes grace à plusieurs autres méthodes :

- doHead() : pour les requêtes http de type HEAD
- doPut() : pour les requêtes http de type PUT
- doDelete() : pour les requêtes http de type DELETE
- doOptions() : pour les requêtes http de type OPTIONS
- doTrace() : pour les requêtes http de type TRACE

La classe HttpServlet hérite aussi de plusieurs méthodes définies dans l'interface Servlet : init(), destroy() et getServletInfo().

35.4.1. La méthode init()

Si cette méthode doit être redéfinie, il est important d'invoquer la méthode héritée avec un appel à super.init(config), config étant l'objet fourni en paramètre de la méthode. Cette méthode définie dans la classe HttpServlet sauvegarde l'objet de type ServletConfig.

De plus, le classe GenericServlet implémente l'interface ServletConfig. Les méthodes redéfinies pour cette interface utilisent l'objet sauvegardé. Ainsi, la servlet peut utiliser sa propre méthode getInitParameter() ou utiliser la méthode getInitParameter() de l'objet de type ServletConfig. La première solution permet un usage plus facile dans toute la servlet.

Sans l'appel à la méthode héritée lors d'une redéfinition, la méthode getInitParameter() de la servlet levera une exception de type NullPointerException.

35.4.2. L'analyse de la requête

La méthode service() est la méthode qui est appelée lors d'un appel à la servlet.

Par défaut dans la classe HttpServlet, cette méthode contient du code qui réalise une analyse de la requête client contenue dans l'objet HttpServletRequest. Selon le type de requête GET ou POST, elle appelle la méthode doGet() ou doPost(). C'est bien ce type de requête qui indique quelle méthode utiliser dans la servlet.

Ainsi, la méthode service() n'est pas à redéfinir pour ces requêtes et il suffit de redéfinir les méthodes doGet() et/ou doPost() selon les besoins.

35.4.3. La méthode doGet()

Une requête de type GET est utile avec des liens. Par exemple :

```
<A HREF="http://localhost:8080/examples/servlet/tomcat1.MyHelloServlet">test de la servlet</A>
```

Dans une servlet de type HttpServlet, une telle requête est associée à la méthode doGet().

La signature de la méthode doGet() :

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException
{

}
```

Le traitement typique de la méthode doGet() est d'analyser les paramètres de la requête, alimenter les données de l'en−tête de la réponse et d'écrire la réponse.

35.4.4. La méthode doPost()

Une requête POST n'est utilisable qu'avec un formulaire HTML.

Exemple de code HTML :

```
<FORM ACTION="http://localhost:8080/examples/servlet/tomcat1.TestPostServlet"
METHOD="POST">
<INPUT NAME="NOM">
<INPUT NAME="PRENOM">
<INPUT TYPE="ENVOYER">
</FORM>
```

Dans l'exemple ci dessus, le formulaire comporte deux zones de saisies correspondant à deux paramètres : NOM et PRENOM.

Dans une servlet de type HttpServlet, une telle requête est associée à la méthode doPost().

La signature de la méthode doPost() :

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException
{

}
```

La méthode doPost() doit généralement recueillir les paramètres pour les traiter et générer la réponse. Pour obtenir la valeur associée à chaque paramètre il faut utiliser la méthode getParameter() de l'objet HttpServletRequest. Cette méthode attends en paramètre le nom du paramètre dont on veut la valeur. Ce paramètre est sensible à la casse.

Exemple :

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException
{
   String nom = request.getParameter("NOM");
   String prenom = request.getParameter("PRENOM");
}
```

35.4.5. La génération de la réponse

La servlet envoie sa réponse au client en utilisant un objet de type HttpServetResponse. HttpServletResponse est une interface : il n'est pas possible d'instancier un tel objet mais le moteur de servlet instancie un objet qui implémente cette interface et le passe en paramètre de la méthode service.

Cette interface possède plusieurs méthodes pour mettre à jour l'en−tête http et le page HTML de retour.

Méthode	Rôle

void sendError (int)	Envoie une erreur avec un code retour et un message par défaut
void sendError (int, String)	Envoie une erreur avec un code retour et un message
void setContentType(String)	Héritée de ServletResponse, cette méthode permet de préciser le type MIME de la réponse
void setContentLength(int)	Héritée de ServletResponse, cette méthode permet de préciser la longueur de la réponse
ServletOutputStream getOutputStream()	Héritée de ServletResponse, elle retourne un flux pour l'envoi de la réponse
PrintWriter getWriter()	Héritée de ServletResponse, elle retourne un flux pour l'envoi de la réponse

Avant de générer la réponse sous forme de page HTML, il faut indiquer dans l'en−tête du message http, le type mime du contenu du message. Ce type sera souvent « text/html » qui correspond à une page HTML mais il peut aussi prendre d'autres valeurs en fonction de ce que retourne la servlet (une image par exemple). La méthode à utiliser est setContentType().

Il est aussi possible de préciser la longueur de la réponse avec la méthode setContentLength(). Cette précision est optionnelle mais si elle est utilisée, la longueur doit être exacte pour éviter des problèmes.

Il est préférable de créer une ou plusieurs méthodes recevant en paramètre l'objet HttpServletResponse qui seront dédiées à la génération du code HTML afin de ne pas alourdir les méthodes doXXX().

Il existe plusieurs façons de générer une page HTML : elles utiliseront toutes soit la méthode getOutputStream() ou getWriter() pour obtenir un flux dans lequel la réponse sera envoyée.

- Utilisation d'un StringBuffer et getOutputStream

Exemple (code Java 1.1) :

```
protected void GenererReponse1(HttpServletResponse reponse) throws IOException
{
   //creation de la reponse
   StringBuffer sb = new StringBuffer();
   sb.append("<HTML>\n");
   sb.append("<HEAD>\n");
   sb.append("<TITLE>Bonjour</TITLE>\n");
   sb.append("</HEAD>\n");
   sb.append("<BODY>\n");
   sb.append("<H1>Bonjour</H1>\n");
   sb.append("</BODY>\n");
   sb.append("</HTML>");

   // envoie des infos de l'en tete
   reponse.setContentType("text/html");
   reponse.setContentLength(sb.length());

   // envoie de la reponse
   reponse.getOutputStream().print(sb.toString());
}
```

L'avantage de cette méthode est qu'elle permet facilement de déterminer la longueur de la réponse.

Dans l'exemple, l'ajout des retours chariot '\n' à la fin de chaque ligne n'est pas obligatoire mais elle facilite la compréhension du code HTML surtout si il devient plus complexe.

- Utilisation directe de getOutputStream

Exemple :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestServlet4 extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();
      out.println("<HTML>\n");
      out.println("<HEAD>\n");
      out.println("<TITLE>Bonjour</TITLE>\n");
      out.println("</HEAD>\n");
      out.println("<BODY>\n");
      out.println("<H1>Bonjour</H1>\n");
      out.println("</BODY>\n");
      out.println("</HTML>");
   }

}
```

- Utilisation de la méthode getWriter()

Exemple (code Java 1.1) :

```
protected void GenererReponse2(HttpServletResponse reponse) throws IOException {

   reponse.setContentType("text/html");

   PrintWriter out = reponse.getWriter();

   out.println("<HTML>");
   out.println("<HEAD>");
   out.println("<TITLE>Bonjour</TITLE>");
   out.println("</HEAD>");
   out.println("<BODY>");
   out.println("<H1>Bonjour</H1>");
   out.println("</BODY>");
   out.println("</HTML>");
}
```

Avec cette méthode, il faut préciser le type MIME avant d'écrire la réponse. L'emploi de la méthode println() permet d'ajouter un retour chariot en fin de chaque ligne.

Si un problème survient lors de la génération de la réponse, la méthode sendError() permet de renvoyer une erreur au client : un code retour est positionné dans l'en−tête http et le message est indiqué dans une simple page HTML.

35.4.6. Un exemple de servlet HTTP très simple

Toute servlet doit au moins importer trois packages : java.io pour la gestion des flux et deux packages de l'API servlet ; javax.servlet.* et javax.servlet.http.

Il faut déclarer une nouvelle classe qui hérite de HttpServlet.

Il faut redéfinir la méthode doGet() pour y insérer le code qui va envoyer dans un flux le code HTML de la page générée.

Exemple (code Java 1.1) :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class MyHelloServlet extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException {

      response.setContentType("text/html");

      PrintWriter out = response.getWriter();

      out.println("<html>");
      out.println("<head>");
      out.println("<title>Bonjour tout le monde</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Bonjour tout le monde</h1>");
      out.println("</body>");
      out.println("</html>");
   }
}
```

La méthode getWriter() de l'objet HttpServletResponse renvoie un flux de type PrintWriter dans lequel on peut écrire la réponse.

Si aucun traitement particulier n'est associé à une requête de type POST, il est pratique de demander dans la méthode doPost() d'exécuter la méthode doGet(). Dans ce cas, la servlet est capable de renvoyer une réponse pour les deux types de requête.

Exemple (code Java 1.1) :

```
public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException {

   this.doGet(request, response);

}
```

35.5. Les informations sur l'environnement d'exécution des servlets

Une servlet est exécutée dans un contexte particulier mis en place par le moteur de servlet.

La servlet peut obtenir des informations sur ce contexte.

La servlet peut aussi obtenir des informations à partir de la requête du client.

35.5.1. Les paramètres d'initialisation

Dès que de la servlet est instanciée, le moteur de servlet appelle sa méthode init() en lui donnant en paramètre un objet de type ServletConfig.

ServletConfig est une interface qui possède deux méthodes permettant de connaître les paramètres d'initialisation :

- String getInitParameter(String) : retourne la valeur du paramètre dont le nom est fourni en paramètre

Exemple :

```
String param;

public void init(ServletConfig config) {

   param = config.getInitParameter("param");
```

```
}
```

- Enumeration getInitParameterNames() : retourne une enumeration des paramètres d'initialisation

Exemple (code Java 1.1) :

```
public void init(ServletConfig config) throws ServletException {

   cfg = config;

   System.out.println("Liste des parametres d'initialisation");

   for (Enumeration e=config.getInitParameterNames(); e.hasMoreElements();) {

      System.out.println(e.nextElement());

   }

}
```

La déclaration des paramètres d'initialisation dépend du serveur qui est utilisé.

35.5.2. L'objet ServletContext

La servlet peut obtenir des informations à partir d'un objet ServletContext retourné par la méthode getServletContext() d'un objet ServletConfig.

Il est important de s'assurer que cet objet ServletConfig, obtenu par la méthode init() est soit explicitement sauvegardé soit sauvegardé par l'appel à la méthode init() héritée qui effectue cette sauvegarde.

L'interface ServletContext contient plusieurs méthodes dont les principales sont :

méthode	Role	Deprecated
String getMimeType(String)	Retourne le type MIME du fichier en paramètre	
String getServletInfo()	Retourne le nom et le numero de version du moteur de servlet	
Servlet getServlet(String)	Retourne une servlet à partir de son nom grace au context	Ne plus utiliser depuis la version 2.1 du jsdk
Enumeration getServletNames()	Retourne une enumeration qui contient la liste des servlets relatives au contexte	Ne plus utiliser depuis la version 2.1 du jsdk
void log(Exception, String)	Ecrit les informations fournies en paramètre dans le fichier log du serveur	Utiliser la nouvelle méthode surchargée log()
void log(String)	Idem	
void log (String, Throwable)	Idem	

Exemple : écriture dans le fichier log du serveur :

```
public void init(ServletConfig config) throws ServletException {

   ServletContext sc = config.getServletContext();

   sc.log( "Demarrage servlet TestServlet" );

}
```

Le format du fichier log est dependant du serveur utilisé :

Exemple : résultat avec tomcat

```
Context log path="/examples" :Demarrage servlet TestServlet
```

35.5.3. Les informations contenues dans la requête

De nombreuses informations en provenance du client peuvent être extraites de l'objet ServletRequest passé en paramètre par le serveur (ou de HttpServletRequest qui hérite de ServletRequest).

Les informations les plus utiles sont les paramètres envoyés dans la requête.

L'interface ServletRequest dispose de nombreuses méthodes pour obtenir ces informations :

Méthode	Role
int getContentLength()	Renvoie la taille de la requête, 0 si elle est inconnue
String getContentType()	Renvoie le type MIME de la requête, null si il est inconnu
ServletInputStream getInputStream()	Renvoie un flux qui contient le corps de la requête
Enumeration getParameterNames()	Renvoie une énumeration contenant le nom de tous les paramètres
String getProtocol()	Retourne le nom du protocole et sa version utilisé par la requête
BufferedReader getReader()	Renvoie un flux qui contient le corps de la requête
String getRemoteAddr()	Renvoie l'adresse IP du client
String getRemoteHost()	Renvoie le nom de la machine cliente
String getScheme	Renvoie le protocole utilisé par la requête (exemple : http, ftp ...)
String getServerName()	Renvoie le nom du serveur qui a reçu la requête
int getServerPort()	Renvoie le port du serveur qui a reçu la requête

Exemple (code Java 1.1) :

```
package tomcat1;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class InfoServlet extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException,ServletException {

      GenererReponse(request, response);

   }

   protected void GenererReponse(HttpServletRequest request, HttpServletResponse reponse)
   throws IOException {

      reponse.setContentType("text/html");

      PrintWriter out =reponse.getWriter();

      out.println("<html>");
```

```
      out.println("<body>");
      out.println("<head>");
      out.println("<title>Informationsa disposition de la servlet</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<p>Typemime de la requête :"
        +request.getContentType()+"</p>");
      out.println("<p>Protocolede la requête :"
        +request.getProtocol()+"</p>");
      out.println("<p>AdresseIP du client :"
        +request.getRemoteAddr()+"</p>");
      out.println("<p>Nom duclient : "
        +request.getRemoteHost()+"</p>");
      out.println("<p>Nom duserveur qui a reçu la requête :"
        +request.getServerName()+"</p>");
      out.println("<p>Port duserveur qui a reçu la requête :"
        +request.getServerPort()+"</p>");
      out.println("<p>scheme: "+request.getScheme()+"</p>");
      out.println("<p>listedes parametres </p>");

      for (Enumeration e =request.getParameterNames() ; e.hasMoreElements() ; ) {

         Object p = e.nextElement();
         out.println("<p>  nom : "+p+" valeur :"
            +request.getParameter(""+p)+"</p>");

      }

      out.println("</body>");
      out.println("</html>");

   }
}
```

Résultat : avec l'url http://localhost:8080/examples/servlet/tomcat1.InfoServlet?param1=valeur1&param2=valeur2 : Une page html s'affiche contenant :

```
Type mime de la requête : null

Protocole de la requête : HTTP/1.0

Adresse IP du client : 127.0.0.1

Nom du client : localhost

Nom du serveur qui a reçu la requête : localhost

Port du serveur qui a reçu la requête : 8080

scheme : http

liste des parametres

   nom : param2 valeur :valeur2

   nom : param1 valeur :valeur1
```

35.6. La mise en oeuvre des servlets avec Tomcat

Pour pouvoir mettre en oeuvre des servlets, il faut obligatoirement utiliser un conteneur web qui propose un environnement d'exécution pour les servlets. Il en existe plusieurs commerciaux ou libres. Le plus connu et le plus utilisé, diffusé selon le modèle open source, est l'implémentation de référence nommée Tomcat et développée par le projet Jakarta du groupe Apache.

Il existe plusieurs versions de Tomcat qui mettent en oeuvre des versions différentes des spécifications des servlets et des JSP :

Version de Tomcat	Version Servlet	Version JSP
3.0, 3.1, 3.2, 3.3	2.2	1.1
4.0, 4.1	2.3	1.2
5.0	2.4	2.0

35.6.1. Installation de Tomcat

Comme Tomcat est écrit en Java, il est possible de l'installer et de l'exécuter sous tous les environnements disposant d'une machine virtuelle Java.

35.6.1.1. L'installation de Tomcat 3.1 sur Windows 98

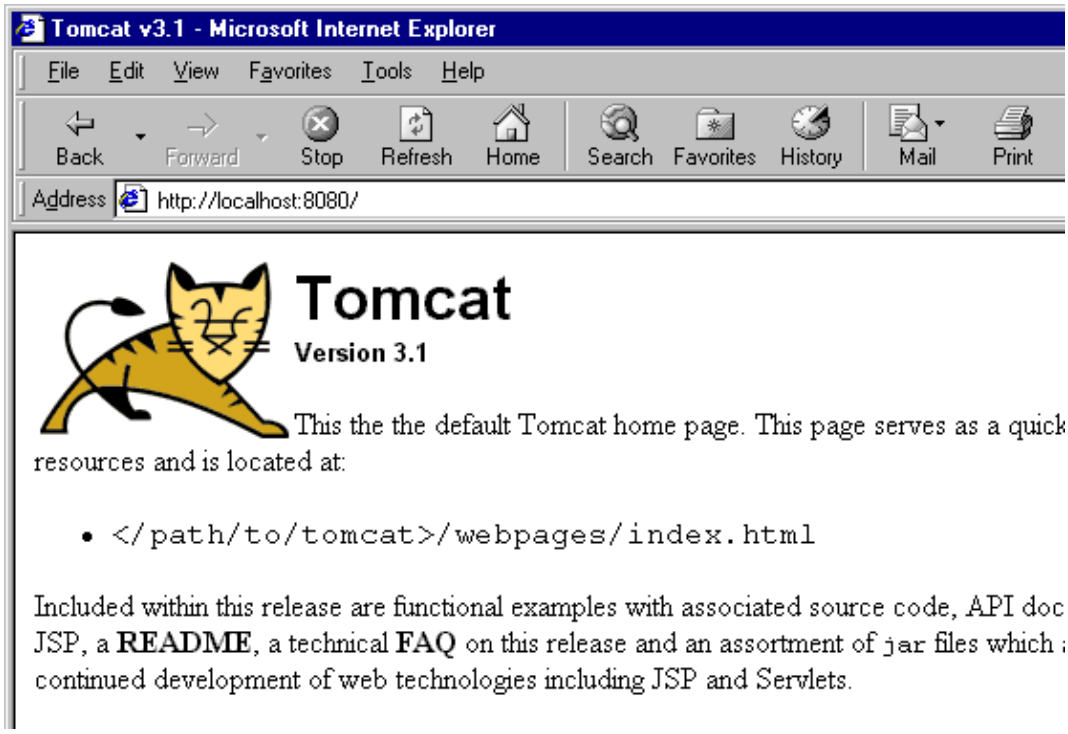
Il est possible de récuperer tomcat sur le site de Jakarta. Il faut choisir la version stable de préférence et le répertoire bin pour récuperer le fichier jakarta−tomcat.zip

Il faut ensuite décompresser le fichier dans un répertoire du système par exemple dans C:\. L'archive est décompressée dans un répertoire nommé jakarta−tomcat

Dans une boite DOS, assigner le répertoire contenant tomcat dans une variable d'environnement TOMCAT_HOME. Le plus simple est de l'ajouter dans le fichier autoexec.bat.
Exemple : set TOMCAT_HOME=c:\jakarta−tomcat

Pour lancer Tomcat, il faut exécuter le fichier startup.bat dans le répertoire TOMCAT_HOME\bin

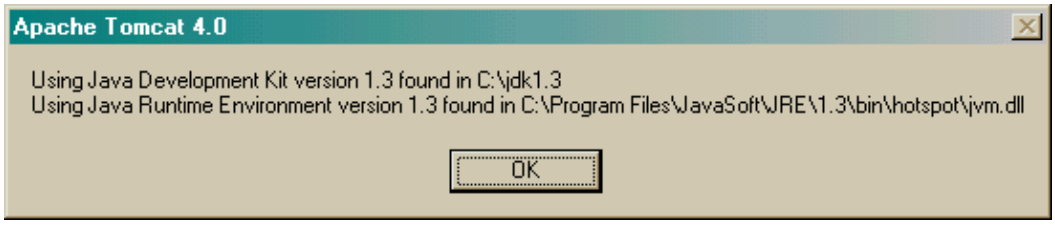
Pour vérifier que Tomcat s'execute correctement, il faut saisir l'url http://localhost:8080 dans un browser. La page d'accueil de Tomcat s'affiche.



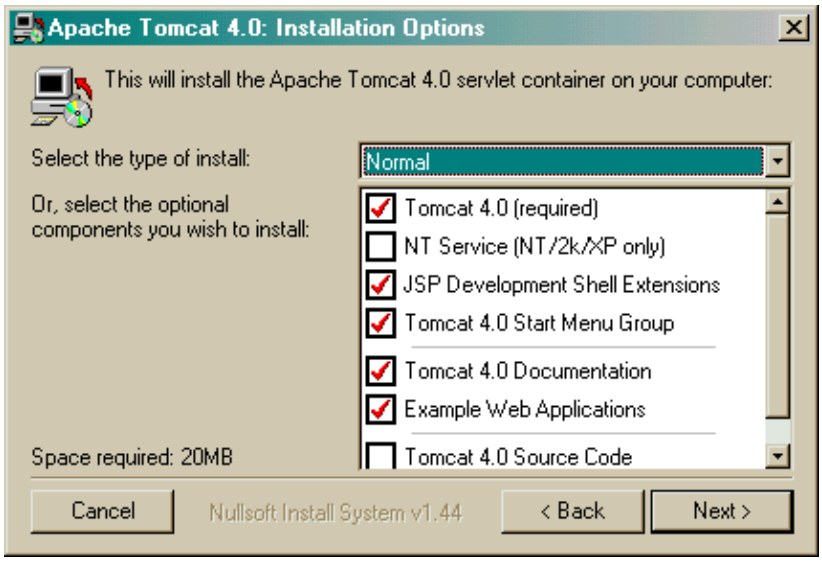
Le script %TOMCAT_HOME%\bin\shutdown.bat permet de stopper Tomcat.

35.6.1.2. L'installation de Tomcat 4.0 sur Windows 98

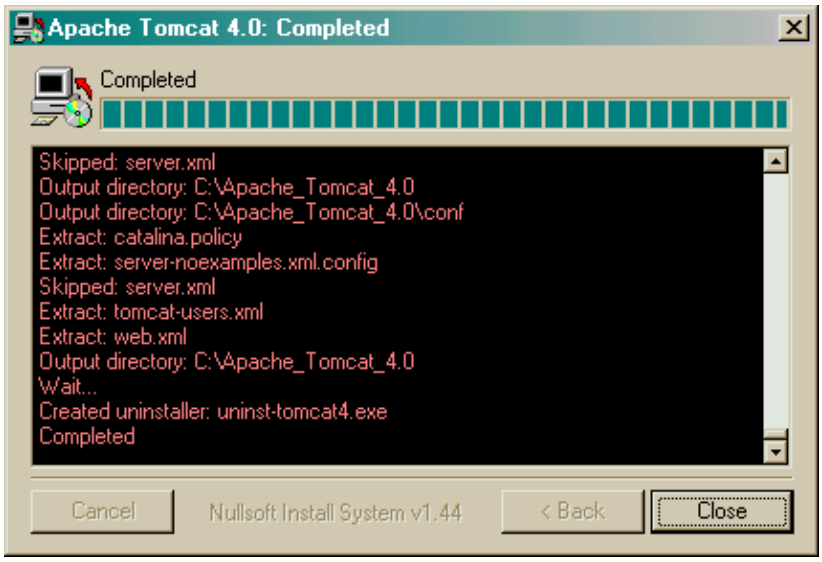
Il suffit de télécharger et d'exécuter le programme jakarta−tomcat−4.0.exe



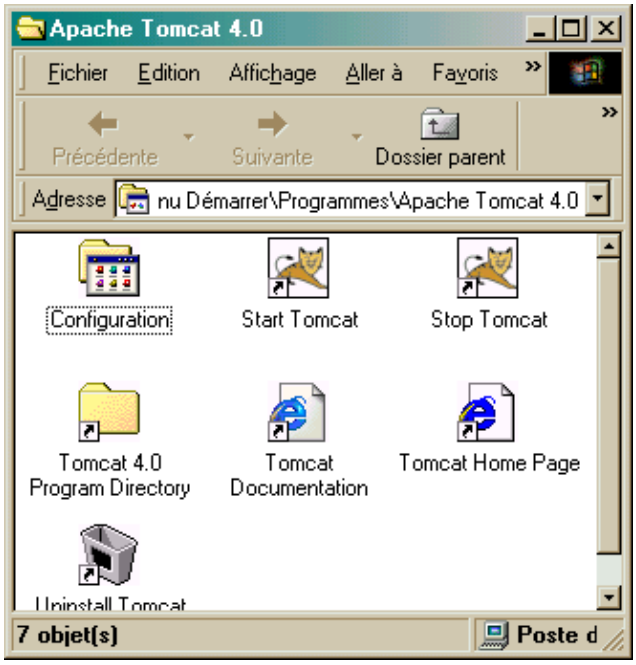
L'assistant affiche la licence, puis permet de sélectionner les options d'installation et le répertoire d'installation.



L'assistant copie les fichiers.



Un ensemble de raccourcis est créé dans l'option "Apache Tomcat 4.0" du menu "Démarrer/Programmes"



Il faut définir la variable d'environnement système JAVA_HOME qui doit avoir comme valeur le chemin absolu du répertoire d'installation du J2SDK.

Pour la version 4.1, il faut télécharger le fichier jakarta-tomcat-4.1.29.exe sur le site http://jakarta.apache.org/site/binindex.cgi

35.6.1.3. L'installation de Tomcat 5.0 sur Windows

Il faut télécharger le fichier jakarta-tomcat-5.0.16.exe sur le site http://jakarta.apache.org/site/binindex.cgi

La version 5 utilise un programme d'installation standard guidé par un assistant qui propose les étapes suivantes :

- la page d'accueil s'affiche, cliquez sur le bouton « Next »
- la page d'acceptation de la licence (« Licence agreement ») s'affiche, lire la licence et si vous l'acceptez, cliquez sur le bouton « I Agree »
- la page de sélection des composants à installer (« Choose components ») s'affiche, il faut sélectionner ou non chacun des composants ou utiliser un type d'installation qui contient une pré configuration, cliquez sur le bouton « Next »
- la page de sélection du répertoire d'installation (« Choose install location ») s'affiche, sélectionner le répertoire de destination et cliquez sur le bouton « Next »
- la page de configuration (« Basic configuration ») s'affiche et permet de définir le port du connecteur http (8080 par défaut) utilisé, le nom et le mot de passe de l'administrateur. Saisissez ces informations et cliquez sur le bouton « Next »
- la page de sélection du chemin de la JVM (« Java virtual machine ») permet de sélectionner le chemin du JRE. Cliquez sur le bouton « Install »
- l'installation s'effectue
- la page de fin d'affiche. Une case à cocher permet de demander le lancement de Tomcat. Cliquez sur le bouton « Finish »

35.6.2. L'utilisation de Tomcat 4.x

Le coeur de Tomcat est composé d'un conteneur Web capable d'exécuter des servlets et par extension des JSP. Tomcat propose aussi des services annexes tel qu'un serveur web.

Pour la réception des requêtes, Tomcat utilisent un connecteur.

Le répertoire où est installé Tomcat est composé de l'arborescence suivante :

- bin : contient un ensemble de scripts pour la mise en oeuvre de Tomcat
- common : le sous répertoire lib contient les bibliothèques utilisées par Tomcat et mises à disposition de toutes les applications qui seront exécutées dans Tomcat
- conf : contient des fichiers de propriétés notamment les fichiers server.xml, tomcat−users.xml et le fichier par défaut web.xml
- logs : contient les journaux d'exécution
- server : contient des bibliothèques utilisées par Tomcat et l'application web d'administration de Tomcat
- temp : est un répertoire temporaire utilisé lors des traitements
- webapps : contient les applications web exécutées sous Tomcat
- work : contient le résultat de la compilation des JSP en servlets

Sous Windows, pour lancer Tomcat manuellement, il faut exécuter la commande startup.bat dans le sous répertoire bin du répertoire où est installé Tomcat. La commande shutdown.bat permet inversement de stopper l'exécution de Tomcat.

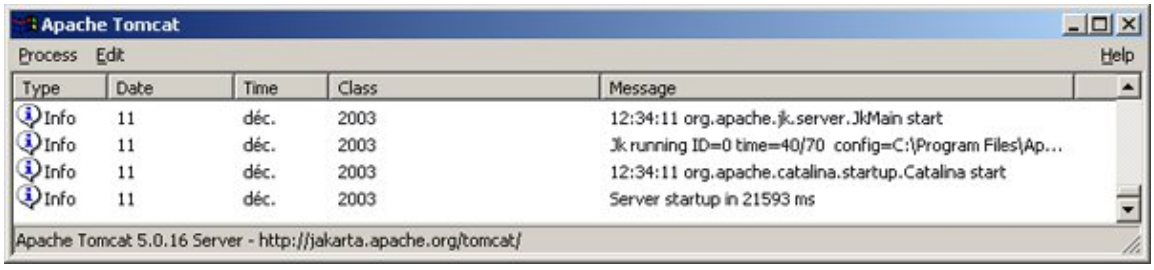
Par défaut, le serveur web intégré dans Tomcat utilise le port 8080 pour recevoir les requêtes HTTP. Pour vérifier la bonne installation de l'outil, il suffit d'ouvrir un navigateur et de demander l'URL : http://localhost:8080/

Le fichier de configuration principale est le fichier server.xml stocké dans le répertoire conf. Il permet notamment de configurer les différents éléments qui composent Tomcat.

35.6.3. L'utilisation de Tomcat 5.x

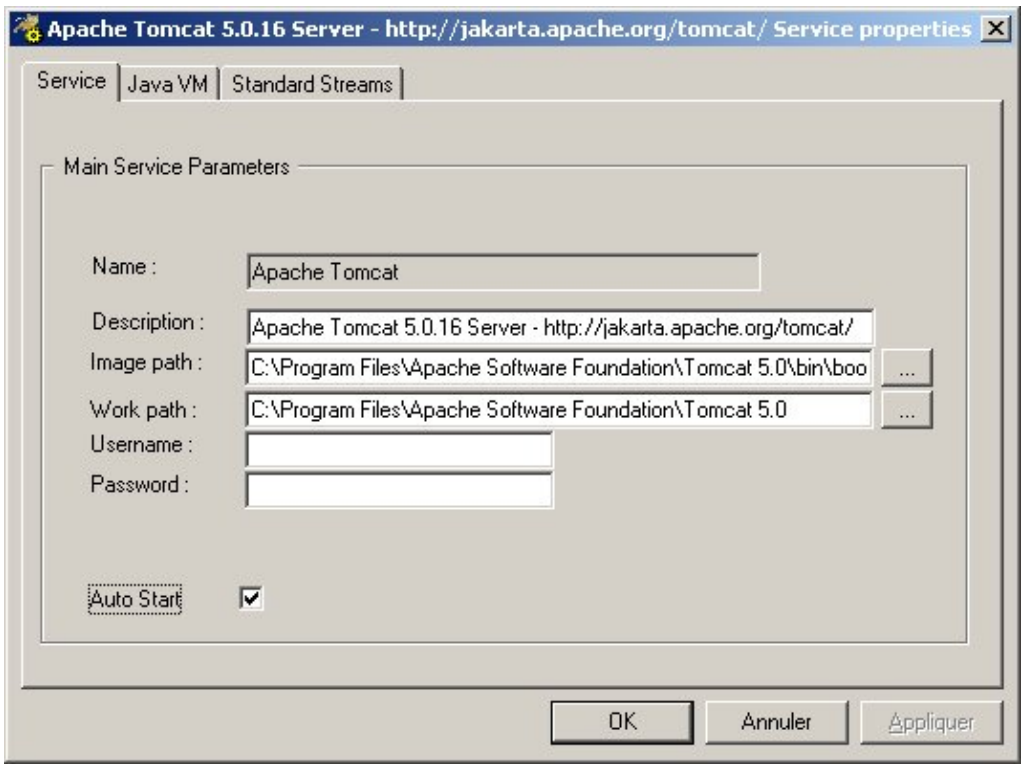
Après le démarrage de Tomcat, une icône apparaît dans la barre  . Elle possède un menu contextuel qui permet de réaliser plusieurs actions :

- « Open Console Monitor » : permet d'afficher un journal des messages émis par Tomcat

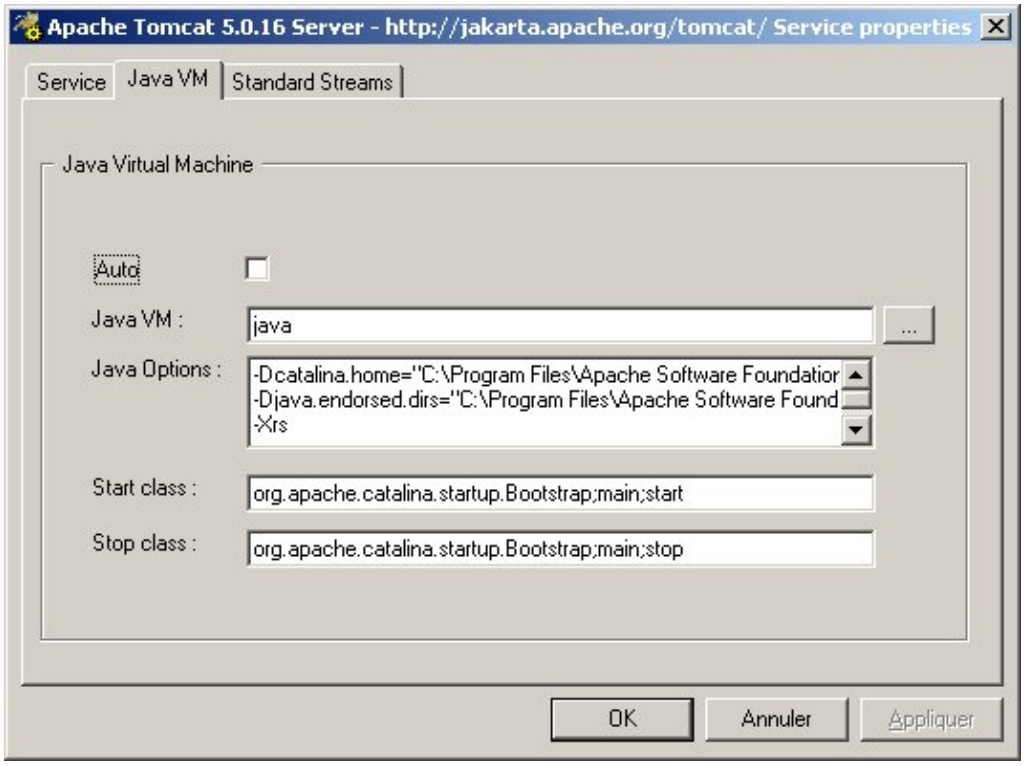


- « About » : permet d'afficher la licence de Tomcat
- « Properties » : permet de changer les propriétés de Tomcat :

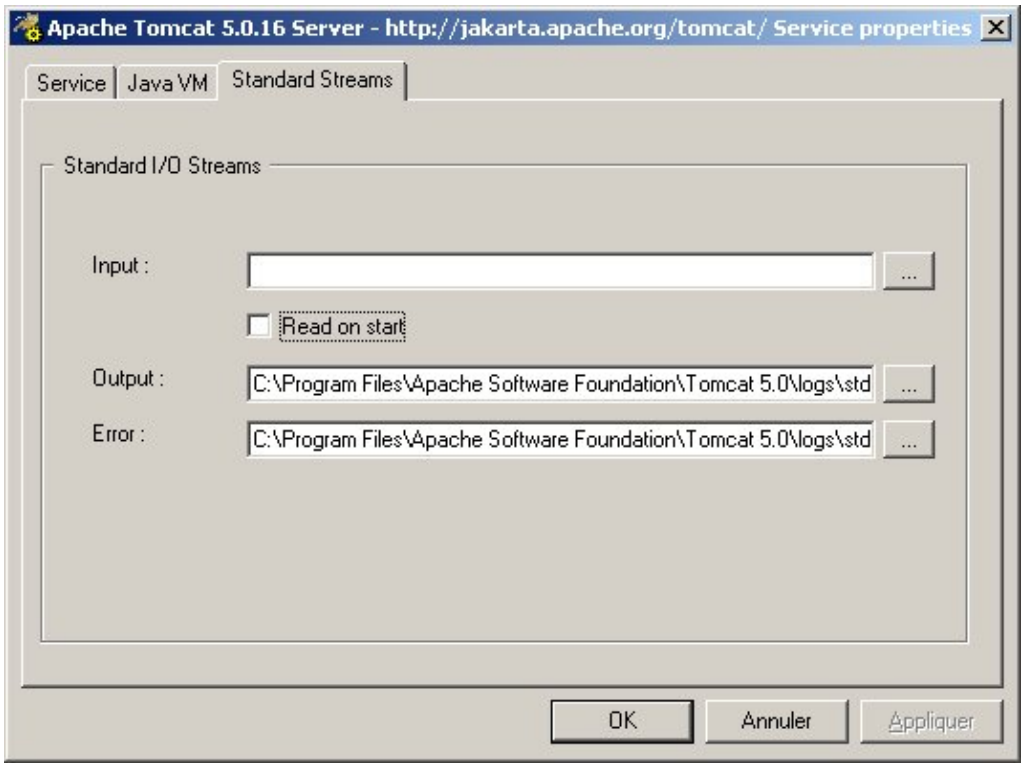
L'onglet « Service » permet de préciser les informations générales concernant l'exécution de Tomcat



L'onglet Java VM permet de préciser les options utilisées lors du lancement de la JVM dans laquelle Tomcat s'exécute



L'onglet « Standard Streams » permet de préciser la localisation des fichiers de logs



- « Shutdown » : permet d'arrêter le service Tomcat

Tomcat 5 met à disposition un certain nombre d'API à toutes les webapp qu'il exécute. Ces API sont dans le répertoire "common/lib" ou "shared/lib" de Tomcat :

- ant.jar (Apache Ant 1.6)
- commons−collections*.jar (Commons Collections 2.1)
- commons−dbcp.jar (Commons DBCP 1.1)
- commons−el.jar (Commons Expression Language 1.0)
- commons−logging−api.jar (Commons Logging API 1.0.3)
- commons−pool.jar (Commons Pool 1.1)

- jasper−compiler.jar
- jsp−api.jar (JSP 2.0)
- commons−el.jar (JSP 2.0 EL)
- naming−common.jar
- naming−factory.jar
- naming−resources.jar
- servlet−api.jar (Servlet 2.4)

35.7. L'utilisation des cookies

Les cookies sont des fichiers contenant des données au format texte, envoyés par le serveur et stockés sur le poste client. Les données contenues dans le cookie sont renvoyées au serveur à chaque requête.

Les cookies peuvent être utilisés explicitement ou implicitement par exemple lors de l'utilisation d'une session.

Les cookies ne sont pas dangereux car ce sont uniquement des fichiers textes qui ne sont pas executés. De plus, les navigateurs posent des limites sur le nombre (en principe 20 cookies pour un même serveur) et la taille des cookies (4ko maximum). Par contre les cookies peuvent contenir des données plus ou moins sensibles. Il est capital de ne stocker dans les cookies que des données qui ne sont pas facilement exploitables par une intervention humaine sur le poste client et tout cas de ne jamais les utiliser pour stocker des informations sensibles tel qu'un numéro de carte bleue.

35.7.1. La classe Cookie

La classe javax.servlet.http.Cookie encapsule un cookie.

Un cookie est composé d'un nom, d'une valeur et d'attributs.

Pour créer un cookie, il suffit d'instancier un nouvel objet de type Cookie. La classe Cookie ne possède qu'un seul constructeur qui attend deux paramètres de type String : le nom et la valeur associée.

Le classe Cookie possède plusieurs getter et setter pour obtenir ou définir des attributs qui sont tous optionnels.

Attribut	Rôle
Comment	Commentaire associé au cookie
Domain	Nom de domaine (partiel ou complet) associé au cookie. Seul les serveurs contenant ce nom de domaine recevront le cookie.
MaxAge	Durée de vie en secondes du cookie. Une fois ce délai expiré, le cookie est détruit sur le poste client par le navigateur. Par défaut la valeur limite la durée de vie du cookie à la durée de vie de l'exécution du navigateur
Name	Nom du cookie
Path	Chemin du cookie. Ce chemin permet de renvoyer le cookie uniquement au serveur dont l'url contient également le chemin. Par défaut, cet attribut contient le chemin de l'url de la servlet. Par exemple, pour que le cookie soit renvoyé à toutes les requêtes du serveur, il suffit d'affecter la valeur "/" à cette attribut.
Secure	Booléen qui précise si le cookie ne doit être envoyé que via une connexion SSL.
Value	Valeur associée au cookie.
Version	Version du protocole utilisé pour gérer le cookie

35.7.2. L'enregistrement et la lecture d'un cookie

Pour envoyer un cookie au browser, il suffit d'utiliser la méthode addCookie() de la classe HttpServletResponse.

Exemple :

```
Cookie monCookie = new Cookie("nom","valeur");
response.addCookie(monCookie);
```

Pour lire un cookie envoyé par le browser, il faut utiliser la méthode getCookies() de la classe HttpServletRequest. Cette méthode renvoie un tableau d'objet Cookie. Les cookies sont renvoyés dans l'en−tête de la requête http. Pour rechercher un cookie particulier, il faut parcourir le tableau et rechercher le cookie à partir de son nom grâce à la méthode getName() de l'objet Cookie.

Exemple :

```
Cookie[] cookies = request.getCookies();
String valeur = "";
for(int i=0;i<cookies.length;i++) {
  if(cookies[i].getName().equals("nom")) {
    valeur=cookies[i].getValue();
  }
}
```

35.8. Le partage d'informations entre plusieurs échanges HTTP

Cette section sera développée dans une version future de ce document

35.9. Packager une application web

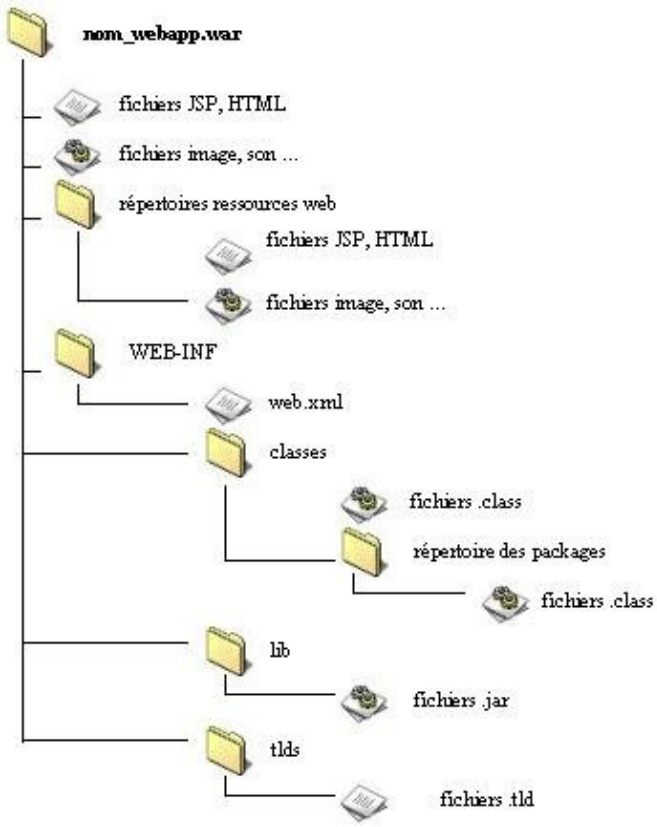
Le format war (Web Application Archive) permet de regrouper en un seul fichier tous les éléments d'une application web que ce soit pour le côté serveur (servlets, JSP, classes java, ...) ou pour le côté client (ressources HTML, images, son ...).

C'est une extension du format jar spécialement dédiée aux applications web qui a été introduite dans les spécifications de la version 2.2 des servlets. C'est un format indépendant de toute plate−forme et exploitable par tous les conteneurs web qui respectent à minima cette version des spécifications.

Le but principal est de simplifier le déploiement d'une application web et d'uniformiser cette action quel que soit le conteneur web utilisé.

35.9.1. Structure d'un fichier .war

Comme les fichiers jar, les fichiers war possèdent une structure particulière qui est incluse dans un fichier compressé de type "zip" possédant comme extension ".war".



Le nom du fichier .war est important car ce nom sera automatiquement associé dans l'url pour l'accès à l'application en concatenant le nom du domaine, un slash et le nom du fichier war. Par exemple, pour un serveur web sur le poste local avec un fichier test.war déployé sur le serveur d'application, l'url pour accéder à l'application web sera http://localhost/test/

Le répertoire WEB−INF et le fichier web.xml qu'il contient doivent obligatoirement être présents dans l'archive. Le fichier web.xml est le descripteur de déploiement de l'application web.

Le serveur web peut avoir accès via le serveur d'application à toutes les ressources contenues dans le fichier .war hormis celles contenues dans le répertoire WEB−INF. Ces dernières ne sont accessibles qu'au serveur d'application.

Le répertoire WEB−INF/classes est automatiquement ajouté par le conteneur au CLASSPATH lors du déploiement de l'application web.

L'archive web peut être créée avec l'outil jar fourni avec le JDK ou avec un outil commercial. Avec l'outil jar, il suffit de créer l'arborescence de l'application, de se placer dans le répertoire racine de cette arborescence et d'exécuter la commande :

jar cvf nom_web_app.war .

Toute l'arborescence avec les fichiers qu'elle contient sera incluse dans le fichier nom_web_app.jar.

35.9.2. Le fichier web.xml

Le fichier /WEB−INF/web.xml est un fichier au format XML qui est le descripteur de déploiement permettant de configurer : l'application, les servlets, les sessions, les bibliothèques de tags personnalisées, les paramètres de contexte, les types Mimes, les pages par défaut, les ressouces externes, la sécurité de l'application et des ressources J2EE.

Le fichier web.xml commence par un prologue et une indication sur la version de la DTD à utiliser. Celle−ci dépend des spécifications de l'API servlet utilisée.

Exemple : servlet 2.2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

Exemple : servlet 2.3

```
 <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

L'élément racine est le tag <web-app>. Cet élément peut avoir plusieurs tags fils dont l'ordre d'utilisation doit respecter celui défini dans la DTD utilisée.

Le tag <icon> permet de préciser une petite et une grande image qui pourront être utilisées par des outils graphiques.

Le tag <display-name> permet de donner un nom pour l'affichage dans les outils.

Le tag <description> permet de fournir un texte de description de l'application web.

Le tag <context-param> permet de fournir un paramètre d'initialisation de l'application. Ce tag peut avoir trois tags fils : <param-name>, <param-value> et <description>. Il doit y en avoir autant que de paramètres d'initialisation. Les valeurs fournies peuvent être retrouvées dans le code de la servlet grace à la méthode getInitParameter() de l'objet ServletContext.

La tag <servlet> permet de définir une servlet. Le tag fils <icon> permet de préciser une petite et une grande image pour les outils graphique. Le tag <servlet-name> permet de donner un nom à la servlet qui sera utilisé pour le mapping avec l'URL par défaut de la servlet. Le tag <display-name> permet de donner un nom d'affichage. Le tag <description> permet de fournir une description de la servlet. Le tag <servlet-class> permet de préciser le nom complètement qualifié de la classe java dont la servlet sera une instance. Le tag <init-param> permet de préciser un paramètre d'initialisation pour la servlet. Ce tag possède les tag fils <param-name>, <param-value>, <description>. Les valeurs fournies peuvent être retrouvées dans le code de la servlet grace à la méthode getInitParameter() de la classe ServletConfig. Le tag <load-on-startup> permet de préciser si la servlet doit être instanciée lors de l'initialisation du conteneur. Il est possible de préciser dans le corps de ce tag un numéro de séquence qui pemettra d'ordonner la création des servlets.

Exemple : servlet 2.2

```
 <servlet>
    <servlet-name>MaServlet</servlet-name>
    <servlet-class>com.moi.test.servlet.MaServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
                  <param-name>param1</param-name>
                        <param-value>valeur1</param-value>
    </init-param>

  </servlet>
```

Le tag <servlet-mapping> permet d'associer la servlet à une URL. Ce tag possède les tag fils <servlet-name> et <servlet-mapping>.

Exemple : servlet 2.2

```
 <servlet-mapping>
    <servlet-name>MaServlet</servlet-name>
          <url-pattern>/test</url-pattern>
  </servlet-mapping>
```

Le tag <session-config> permet de configurer les sessions. Le tag fils <session-timeout> permet de préciser la durée maximum d'inactivité de la session avant sa destruction. La valeur fournie dans le corps de ce tag est exprimé en minutes.

Exemple : servlet 2.2

```
<session-config>
   <session-timeout>15</session-timeout>
       </session-config>
```

Le tag <mime-mapping> permet d'associer des extensions à un type mime particulier.

Le tag <welcome-file-list> permet de définir les pages par défaut. Chacun des fichiers est défini grace au tag fils <welcome-file>

Exemple : servlet 2.2

```
<welcome-file-list>
   <welcome-file>index.jsp</welcome-file>
   <welcome-file>index.htm</welcome-file>
 <welcome-file-list>
```

Le tag <error-page> permet d'associer une page web à un code d'erreur HTTP particulier ou a une exception java particulière. Le code erreur est précisé avec le tag fils <error-code>. L'exception Java est précisée avec le tag fils <exception-type>. La page web est précisée avec le tag fils <location>.

Le tag <tag-lib> permet de définir une bibliothèque de tags personnalisée. Le tag fils <taglib-uri> permet de préciser l'URI de la bibliothèque. Le tag fils <taglib-location> permet de préciser le chemin de la bibliothèque.

Exemple : déclaration de la biliothèque core de JSTL

```
<taglib>
         <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
         <taglib-location>/WEB-INF/tld/c.tld</taglib-location>
 </taglib>
```

35.9.3. Le déploiement d'une application web

Le déploiement d'une archive web dans un serveur d'application est très facile car il suffit simplement de copier le fichier .war dans le répertoire par défaut dédié aux applications web. Par exemple dans Tomcat, c'est le répertoire webapps. Attention cependant, si chaque conteneur qui respecte les spécifications 1.1 des JSP sait utiliser un fichier .war, leur exploitation par chaque conteneur est légèrement différente.

Par exmple avec Tomcat, il est possible de travailler directement dans le répertoire webapps avec le contenu de l'archive web décompressé. Cette fonctionnalité est particulièrement intéressante lors de la phase de développement de l'application car il n'est alors pas obligatoire de générer l'archive web à chaque modification pour réaliser des tests. Attention, si l'application est redéployée sous la forme d'une archive .war, il faut obligatoirement supprimer le répertoire qui contient l'ancienne version de l'application.

35.10. Utiliser Log4J dans une servlet

Log4J est un framework dont le but est de faciliter la mise en oeuvre de fonctionnalités de logging dans une application. Il est notamment possible de l'utiliser dans une application web. Pour plus de détails sur cette API, consultez le chapitre qui lui est dédié dans ce didacticiel.

Pour utiliser Log4J dans une application web, il est nécessaire d'initialiser Log4J avant utilisation. Le plus simple est d'écrire une servlet qui va réaliser cette initialisation et qui sera chargée automatiquement au chargement de l'application web.

Dans la méthode init() de la servlet, deux paramètres sont récupérés et sont utilisés pour

- définir une variable d'environnement qui sera utilisée par Log4J dans son fichier de configuration pour définir le chemin du fichier journal utilisé
- initialiser Log4J en utilisant un fichier de configuration

Exemple :

```
package com.jmd.test.log4j;

import org.apache.log4j.PropertyConfigurator;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;

public class InitServlet extends HttpServlet {
        public void init() {

                String cheminWebApp = getServletContext().getRealPath("/");
                String cheminLogConfig = cheminWebApp + getInitParameter("log4j-fichier-config");
                String cheminLog = cheminWebApp + getInitParameter("log4j-chemin-log");

                File logPathDir = new File( cheminLog );
                System.setProperty( "log.chemin", cheminLog );

                if (cheminLogConfig != null) {
                        PropertyConfigurator.configure(cheminLogConfig);
                }
        }

        public void doGet(HttpServletRequest req, HttpServletResponse res) {
        }
}
```

Dans le fichier web.xml, il faut configurer les servlets utilisées et notamment la servlet définie pour initialiser Log4J. Celle ci attend au moins deux paramètres :

- log4j-fichier-config : ce paramètre doit avoir comme valeur le chemin relatif du fichier de configuration de Log4J par rapport à la racine de l'application web
- log4j-chemin-log : ce paramètre doit avoir comme valeur le chemin du répertoire qui va contenir les fichiers journaux générés par Log4J par rapport à la racine de l'application web

Exemple : le fichier web.xml

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>initservlet</servlet-name>
    <servlet-class>com.jmd.test.log4j.InitServlet</servlet-class>
    <init-param>
      <param-name>log4j-fichier-config</param-name>
      <param-value>WEB-INF/classes/log4j.properties</param-value>
    </init-param>
    <init-param>
      <param-name>log4j-chemin-log</param-name>
      <param-value>WEB-INF/log</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>com.jmd.test.log4j.TestServlet</servlet-class>
  </servlet>

  <servlet-mapping>
```

```
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/test</url-pattern>
  </servlet-mapping>
</web-app>
```

Il est important de demander le chargement automatique de la servlet en donnant la valeur 1 au tag <load-on-startup> de la servlet.

Il faut définir le fichier de configuration nommé par exemple log4j.properties et le placer dans le répertoire WEB-INF/classes de l'application.

Exemple : le fichier log4j.properties

```
# initialisation de la racine du logger avec le niveau INFO
log4j.rootLogger=INFO, A1

# utilisation d'un fichier pour stocker les informations du journal
log4j.appender.A1=org.apache.log4j.FileAppender
log4j.appender.A1.file=${log.chemin}/application.log

# utilisation du layout de base
log4j.appender.A1.layout=org.apache.log4j.SimpleLayout
```

L'utilisation de Log4J dans une servlet est alors équivalente à celle d'une application standalone.

Exemple : une servlet qui utilise Log4J

```
package com.jmd.test.log4j;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Logger;

public class TestServlet extends HttpServlet {

  private static final Logger logger = Logger.getLogger(TestServlet.class);

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    logger.info("initialisation de la servet TestServlet");
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res) {
    StringBuffer sb = new StringBuffer();

    logger.debug("appel doGet de la servlet TestServlet");

    sb.append("<HTML>\n");
    sb.append("<HEAD>\n");
    sb.append("<TITLE>Bonjour</TITLE>\n");
    sb.append("</HEAD>\n");
    sb.append("<BODY>\n");
    sb.append("<H1>Bonjour</H1>\n");
    sb.append("</BODY>\n");
    sb.append("</HTML>");

    res.setContentType("text/html");
    res.setContentLength(sb.length());

    try {
      res.getOutputStream().print(sb.toString());
    } catch (IOException e) {
      e.printStackTrace();
    }
```

```
  }
}
```

Lors de l'exécution de l'application web, le journal est rempli dans le fichier /WEB−INF/log/application.log.

36. Les JSP (Java Servers Pages)

Les JSP (Java Server Pages) sont une technologie Java qui permettent la génération de pages web dynamiques.

La technologie JSP permet de séparer la présentation sous forme de code HTML et les traitements sous formes de classes Java définissant un bean ou une servlet. Ceci est d'autant plus facile que les JSP définissent une syntaxe particulière permettant d'appeler un bean et d'insérer le résultat de son traitement dans la page HTML dynamiquement.

Les informations fournies dans ce chapitre concernent les spécifications 1.0 et ultérieures des JSP.

Ce chapitre contient plusieurs sections :

36.1. Présentation des JSP

Les JSP permettent d'introduire du code Java dans des tags prédéfinis à l'intérieur d'une page HTML. La technologie JSP mélange la puissance de Java côté serveur et la facilité de mise en page d'HTML côté client.

Sun fourni de nombreuses informations sur la technologie JSP à l'adresse suivante : http://java.sun.com/products/jsp/index.html

Une JSP est habituellement constituée :

- de données et de tags HTML
- de tags JSP
- de scriptlets (code Java intégré à la JSP)

Les fichiers JSP possèdent par convention l'extension .jsp.

Concrètement, les JSP sont basées sur les servlets. Au premier appel de la page JSP, le moteur de JSP génère et compile automatiquement une servlet qui permet la génération de la page web. Le code HTML est repris intégralement dans la servlet. Le code Java est inséré dans la servlet.

La servlet générée est compilée et sauvegardée puis elle est exécutée. Les appels suivants de la JSP sont beaucoup plus rapides car la servlet, conservée par le serveur, est directement exécutée.

Il y a plusieurs manières de combiner les technologies JSP, les beans/EJB et les servlets en fonction des besoins pour développer des applications web.

Comme le code de la servlet est généré dynamiquement, les JSP sont relativement difficiles à déboguer.

Cette approche possède plusieurs avantages :

- l'utilisation de Java par les JSP permet une indépendance de la plate−forme d'exécution mais aussi du serveur web utilisé.
- la séparation des traitements et de la présentation : la page web peut être écrite par un designer et les tags Java peuvent être ajoutés ensuite par le développeur. Les traitements peuvent être réalisés par des composants réutilisables (des Java beans).
- les JSP sont basées sur les servlets : tous ce qui est fait par une servlet pour la génération de pages dynamiques peut être fait avec une JSP.

Il existe plusieurs versions des spécifications JSP :

Version	
0.91	Première release
1.0	Juin 1999 : première version finale
1.1	Décembre 1999 :
1.2	Octobre 2000, JSR 053
2.0	JSR 152

36.1.1. Le choix entre JSP et Servlets

Les servlets et les JSP ont de nombreux points communs puisque qu'une JSP est finalement convertie en une servlet. Le choix d'utiliser l'une ou l'autre de ces technologies ou les deux doit être fait pour tirer le meilleur parti de leurs avantages.

Dans une servlet, les traitements et la présentation sont regroupés. L'aspect présentation est dans ce cas pénible à développer et à maintenir à cause de l'utilisation répétitive de méthodes pour insérer le code HTML dans le flux de sortie. De plus, une simple petite modification dans le code HTML nécessite la recompilation de la servlet. Avec un JSP, la séparation des traitements et de la présentation rend ceci très facile et automatique.

Il est préférable d'utiliser les JSP pour générer des pages web dynamiques.

L'usage des servlets est obligatoire si celles ci doivent communiquer directement avec une applet ou une application et non plus avec un serveur web.

36.1.2. JSP et les technologies concurrentes

Il existe plusieurs technologies dont le but est similaire aux JSP notamment ASP, PHP et ASP.Net. Chacunes de ces technologies possèdent des avantages et des inconvénients dont voici une liste non exhaustive.

	JSP	PHP	ASP	ASP.Net
langage	Java	PHP	VBScript ou JScript	Tous les langages supportés par .Net (C#, VB.Net, Delphi, ...)
mode d'exécution	Compilé en pseudo code (byte code)	Interprété	Interprété	Compilé en pseudo code (MSIL)
principaux avantages	Repose sur la plate−forme Java dont elle hérite des avantages	Open source Nombreuses bibliothèques et sources d'applications libres disponibles	Facile à mettre en oeuvre	Repose sur la plate−forme .Net dont elle hérite des avantages Wysiwyg et

		Facile à mettre en oeuvre		evénementiel Code behind pour séparation affichage / traitements
principaux inconvénients	Débogage assez fastdieux Beaucoup de code à écrire	Débogage assez fastdieux Beaucoup de code à écrire support partiel de la POO en attendant la version 5	Débogage assez fastdieux Beaucoup de code à écrire Fonctionne essentiellement sur plate−formes Windows Pas de POO, objet métier encapsulé dans des objets COM lourd à mettre en oeuvre	Fonctionne essentiellement sur plate−formes Windows. (Voir le projet Mono pour le support d'autres plate−formes)

36.2. Les outils nécessaires

Dans un premier temps, Sun a fourni un kit de développement pour les JSP : le Java Server Web Development Kit (JSWDK). Actuellement, Sun a chargé le projet Apache de développer l'implémentation officielle d'un moteur de JSP. Ce projet se nomme Tomcat.

En fonction des versions des API utilisées, il faut choisir un produit différent. Le tableau ci dessous résume le produit a utiliser en fonction de la version des API mise en oeuvre.

Produit	Version	Version de l'API servlet implémentée	Version de l'API JSP implémentée
JSWDK	1.0.1	2.1	1.0
Tomcat	3.2	2.2	1.1
Tomcat	4.0	2.3	1.2
Tomcat	5.0	2.4	2.0

Ces produits sont librement téléchargeables sur le site de Sun à l'adresse suivante :
http://java.sun.com/products/jsp/download.html

Pour télécharger le JSWDK, il faut cliquer sur le lien " archive ".

Il est aussi possible d'utiliser n'importe quel conteneur web compatible avec les spécifications de la plate−forme J2EE. Une liste non exhaustive est fournie dans le chapitre "Les outils libres et commerciaux".

36.2.1. JavaServer Web Development Kit (JSWDK) sous Windows

Le JSWDK est proposé sous la forme d'un fichier zip nommé jswdk_1_0_1−win.zip.

Pour l'installer, il suffit de décompresser l'archive dans un répertoire du système. Pour lancer le serveur, il suffit d'exécuter le fichier startserver.bat

Pour lancer le serveur :

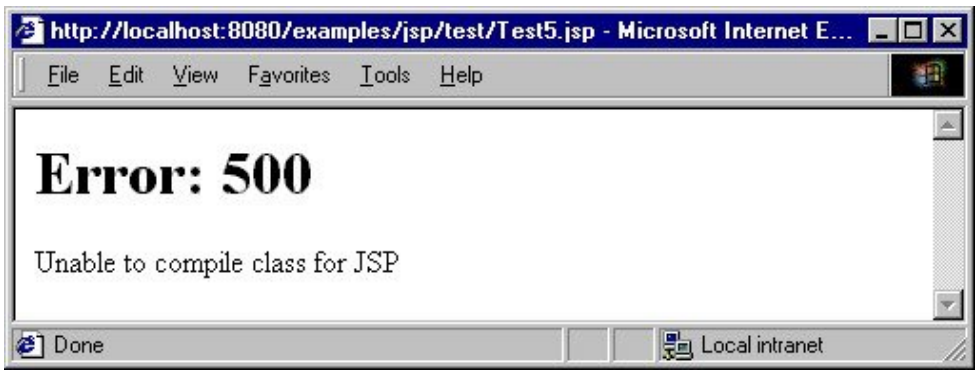
```
C:\jswdk-1.0.1>startserver.bat

Using classpath:.\classes;.\webserver.jar;.\lib\jakarta.jar;.\lib\servlet.jar;.
\lib\jsp.jar;.\lib\jspengine.jar;.\examples\WEB-INF\jsp\beans;.\webpages\WEB-INF
\servlets;.\webpages\WEB-INF\jsp\beans;.\lib\xml.jar;.\lib\moo.jar;\lib\tools.ja
r;C:\jdk1.3\lib\tools.jar;
C:\jswdk-1.0.1>
```

Le serveur s'exécute dans une console en tache de fond. Cette console permet de voir les messages émis par le serveur.

Exemple : au démarrage

```
JSWDK WebServer Version 1.0.1
Loaded configuration from: file:C:\jswdk-1.0.1\webserver.xml
endpoint created: localhost/127.0.0.1:8080
```

Si la JSP contient une erreur, le serveur envoie une page d'erreur :



Une exception est levée et est affichée dans la fenêtre ou le serveur s'exécute :

Exemple :

```
-- Commentaires de la page JSP --
^
1 error
at com.sun.jsp.compiler.Main.compile(Main.java:347)
at com.sun.jsp.runtime.JspLoader.loadJSP(JspLoader.java:135)
at com.sun.jsp.runtime.JspServlet$JspServletWrapper.loadIfNecessary(JspS
ervlet.java:77)
at com.sun.jsp.runtime.JspServlet$JspServletWrapper.service(JspServlet.j
ava:87)
at com.sun.jsp.runtime.JspServlet.serviceJspFile(JspServlet.java:218)
at com.sun.jsp.runtime.JspServlet.service(JspServlet.java:294)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:840)
at com.sun.web.core.ServletWrapper.handleRequest(ServletWrapper.java:155
)
at com.sun.web.core.Context.handleRequest(Context.java:414)
at com.sun.web.server.ConnectionHandler.run(ConnectionHandler.java:139)
HANDLER THREAD PROBLEM: java.io.IOException: Socket Closed
java.io.IOException: Socket Closed
at java.net.PlainSocketImpl.getInputStream(Unknown Source)
at java.net.Socket$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.Socket.getInputStream(Unknown Source)
at com.sun.web.server.ConnectionHandler.run(ConnectionHandler.java:161)
```

Le répertoire work contient le code et le byte code des servlets générées à partir des JSP.

Pour arrêter le serveur, il suffit d'exécuter le script stopserver.bat.

A l'arrêt du serveur, le répertoire work qui contient les servlets générées à partir des JSP est supprimé.

36.2.2. Tomcat

La mise en oeuvre et l'utilisation de Tomcat est détaillée dans une section du chapitre consacré aux servlets.

36.3. Le code HTML

Une grande partie du contenu d'une JSP est constituée de code HTML. D'ailleurs, le plus simple pour écrire une JSP est d'écrire le fichier HTML avec un outil dédié et d'ajouter ensuite les tags JSP pour ce qui concerne les parties dynamiques.

La seule restriction concernant le code HTML concerne l'utilisation dans la page générée du texte " <% " et " %> ". Dans ce cas, le plus simple est d'utiliser les caractères spéciaux HTML &lt ; et &gt ;. Sinon l'analyseur syntaxique du moteur de JSP considère que c'est un tag JSP et renvoie une erreur.

Exemple :

```
<HTML>
<HEAD>
<TITLE>Test</TITLE>
</HEAD>
<BODY>
<p>Plusieurs tags JSP commencent par &lt;% et se finissent par %&gt;</p>
</BODY>
</HTML>
```

36.4. Les Tags JSP

Il existe trois types de tags :

- tags de directives : ils permettent de contrôler la structure de la servlet générée
- tags de scripting: ils permettent d'insérer du code Java dans la servlet
- tags d'actions: ils facilitent l'utilisation de composants

Attention : Les noms des tags sont sensibles à la casse.

36.4.1. Les tags de directives <%@ ... %>

Les directives permettent de préciser des informations globales sur la page JSP. Les spécifications des JSP définissent trois directives :

- page : permet de définir des options de configuration
- include : permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet
- taglib : permet de définir des tags personnalisés

Leur syntaxe est la suivante :

<%@ *directive attribut*="*valeur*" ... %>

36.4.1.1. La directive page

Cette directive doit être utilisée dans toutes les pages JSP : elle permet de définir des options qui s'appliquent à toute la JSP.

Elle peut être placée n'importe ou dans le source mais il est préférable de la mettre en début de fichier, avant même le tag <HTML>. Elle peut être utilisée plusieurs fois dans une même page mais elle ne doit définir la valeur d'une option qu'une seule fois, sauf pour l'option import.

Les options définies par cette directive sont de la forme option=valeur.

Option	Valeur	Valeur par défaut	Autre valeur possible
autoFlush	Une chaîne	«true»	«false»
buffer	Une chaîne	«8kb»	«none» ou «nnnkb» (nnn indiquant la valeur)
contentType	Une chaîne contenant le type mime		
errorPage	Une chaîne contenant une URL		
extends	Une classe		
import	Une classe ou un package.*		
info	Une chaîne		
isErrorPage	Une chaîne	«false»	«true»
isThreadSafe	Une chaîne	«true»	«false»
langage	Une chaîne	«java»	
session	Une chaîne	«true»	«false»

Exemple :

```
<%@ page import="java.util.*" %>
<%@ page import="java.util.Vector" %>
<%@ page info="Ma premiere JSP"%>
```

Les options sont :

- autoFlush="**true**|false"

  Cette option indique si le flux en sortie de la servlet doit être vidé quand le tampon est plein. Si la valeur est false, une exception est levée dès que le tampon est plein. On ne peut pas mettre false si la valeur de buffer est none.

- buffer="none|**8kb**|*size*kb"

  Cette option permet de préciser la taille du buffer des données générées contenues par l'objet out de type JspWriter.

- contentType="*mimeType* [; charset=*characterSet*]" | "**text/html;charset=ISO−8859−1**"

  Cette option permet de préciser le type MIME des données générées.

  Cette option est équivalente à <% response.setContentType("mimeType"); %>

- errorPage=*"relativeURL"*

  Cette option permet de préciser la JSP appelée au cas où une exception est levée

  Si l'URL commence pas un '/', alors l'URL est relative au répertoire principale du serveur web sinon elle est relative au répertoire qui contient la JSP

- extends=*"package.class"*

  Cette option permet de préciser la classe qui sera la super classe de l'objet Java créé à partir de la JSP.

- import= "{ *package.class* | *package.** }, ..."

  Cette option permet d'importer des classes contenues dans des packages utilisées dans le code de la JSP. Cette option s'utilise comme l'instruction import dans un source Java.

  Chaque classe ou package est séparée par une virgule.

  Cette option peut être présente dans plusieurs directives page.

- info="text"

  Cette option permet de préciser un petit descriptif de la JSP. Le texte fourni sera renvoyé par la méthode getServletInfo() de la servlet générée.

- isErrorPage="true|**false**"

  Cette option permet de préciser si la JSP génère une page d'erreur. La valeur true permet d'utiliser l'objet Exception dans la JSP

- isThreadSafe="**true**|false"

  Cette option indique si la servlet générée sera multithread : dans ce cas, une même instance de la servlet peut gérer plusieurs requêtes simultanément. En contre partie, elle doit gérer correctement les accès concurrents aux ressources. La valeur false impose à la servlet générée d'implémenter l'interface SingleThreadModel.

- language="java"

  Cette option définit le langage utilisé pour écrire le code dans la JSP. La seule valeur autorisée actuellement est «java».

- session="**true**|false**"**

  Cette option permet de préciser si la JSP est incluse dans une session ou non. La valeur par défaut (true) permet l'utilisation d'un objet session de type HttpSession qui permet de gérer des informations dans une session.

36.4.1.2. La directive include

Cette directive permet d'inclure un fichier dans le source JSP. Le fichier inclus peut être un fragment de code JSP, HTML ou Java. Le fichier est inclus dans la JSP avant que celle ci ne soit interprétée par le moteur de JSP.

Ce tag est particulièrement utile pour insérer un élément commun à plusieurs pages tel qu'un en−tête ou un bas de page.

Si le fichier inclus est un fichier HTML, celui ci ne doit pas contenir de tag <HTML>, </HTML>, <BODY> ou </BODY> qui ferait double emploi avec ceux présents dans le fichier JSP. Ceci impose d'écrire des fichiers HTML particuliers uniquement pour être inclus dans les JSP : ils ne pourront pas être utilisés seuls.

La syntaxe est la suivante :

<%@ include file="chemin relatif du fichier" %>

Si le chemin commence par un '/', alors le chemin est relatif au contexte de l'application, sinon il est relatif au fichier JSP.

Exemple :

```
bonjour.htm :

<p><table border="1" cellpadding="4" cellspacing="0" width="30%" align=center >
<tr bgcolor="#A6A5C2">
<td align="center">BONJOUR</Td>
</Tr>
</table></p>

Test1.jsp :

<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<p align="center">Test d'inclusion d'un fichier dans la JSP</p>
<%@ include file="bonjour.htm"%>
<p align="center">fin</p>
</BODY>
</HTML>
```

Pour tester cette JSP avec le JSWDK, il suffit de placer ces deux fichiers dans le répertoire jswdk−1.0.1\examples\jsp\test.

Pour visualiser la JSP, il faut saisir l'url http://localhost:8080/examples/jsp/test/Test1.jsp dans un navigateur.

Attention : un changement dans le fichier inclus ne provoque pas une regénération et une compilation de la servlet correspondant à la JSP. Pour insérer un fichier dynamiquement à l'exécution de la servlet il faut utiliser le tag <jsp:include>.

36.4.1.3. La directive taglib

Cette directive permet de déclarer l'utilisation d'une bibliothèque de tags personnalisés. L'utilisation de cette directive est détaillée dans la section consacrée aux bibliothèques de tags personnalisés.

36.4.2. Les tags de scripting

Ces tags permettent d'insérer du code Java qui sera inclus dans la servlet générée à partir de la JSP. Il existe trois tags pour insérer du code Java :

- le tag de déclaration : le code Java est inclus dans le corps de la servlet générée. Ce code peut être la déclaration de variables d'instances ou de classes ou la déclaration de méthodes.
- le tag d'expression : évalue une expression et insère le résultat sous forme de chaîne de caractères dans la page web générée.
- le tag de scriptlets : par défaut, le code Java est inclus dans la méthode service() de la servlet.

Il est possible d'utiliser dans ces tags plusieurs objets définis par les JSP.

36.4.2.1. Le tag de déclarations <%! ... %>

Ce tag permet de déclarer des variables ou des méthodes qui pourront être utilisées dans la JSP. Il ne génère aucun caractère dans le fichier HTML de sortie.

La syntaxe est la suivante :

<%! declarations %>

Exemple :

```
<%! int i = 0; %>
<%! dateDuJour = new java.util.Date(); %>
```

Les variables ainsi déclarées peuvent être utilisées dans les tags d'expressions et de scriptlets.

Il est possible de déclarer plusieurs variables dans le même tag en les séparant avec des caractères ' ;'.

Ce tag permet aussi d'insérer des méthodes dans le corps de la servlet.

Exemple :

```
<HTML>
<HEAD>
<TITLE>Test</TITLE>
</HEAD>
<BODY>
<%!
int minimum(int val1, int val2) {
   if (val1 < val2) return val1;
   else return val2;
}
%>
<% int petit = minimum(5,3);%>
<p>Le plus petit de 5 et 3 est <%= petit %></p>
</BODY>
</HTML>
```

36.4.2.2. Le tag d'expressions <%= ... %>

Le moteur de JSP remplace ce tag par le résultat de l'évaluation de l'expression présente dans le tag.

Ce résultat est toujours converti en une chaîne. Ce tag est un raccourci pour éviter de faire appel à la méthode println() lors de l'insertion de données dynamiques dans le fichier HTML.

La syntaxe est la suivante :

<%= expression %>

Le signe '=' doit être collé au signe '%'.

Attention : il ne faut pas mettre de ' ;' à la fin de l'expression.

Exemple : Insertion de la date dans la page HTML

```
<%@ page import="java.util.*" %>
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<p align="center">Date du jour :
<%= new Date() %>
</p>
</BODY>
</HTML>
```

Résultat :

```
Date du jour : Thu Feb 15 11:15:24 CET 2001
```

L'expression est évaluée et convertie en chaîne avec un appel à la méthode toString(). Cette chaîne est insérée dans la

page HTML en remplacement du tag. Il est ainsi possible que le résultat soit une partie ou la totalité d'un tag HTML ou même une JSP.

Exemple :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<%="<H1>" %>
Bonjour
<%="</H1>" %>
</BODY>
</HTML>
```

Résultat : code HTML généré

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<H1>
Bonjour
</H1>
</BODY>
</HTML>
```

36.4.2.3. Les variables implicites

Les spécifications des JSP définissent plusieurs objets utilisables dans le code dont les plus utiles sont :

Object	Classe	Rôle
out	javax.servlet.jsp.JspWriter	Flux en sortie de la page HTML générée
request	javax.servlet.http.HttpServletRequest	Contient les informations de la requête
response	javax.servlet.http.HttpServletRequest	Contient les informations de la réponse
session	javax.servlet.http.HttpSession	Gère la session

36.4.2.4. Le tag des scriptlets <% ... %>

Ce tag contient du code Java nommé un scriptlet.

La syntaxe est la suivante : <% code Java %>

Exemple :

```
<%@ page import="java.util.Date"%>
<html>
<body>
<%! Date dateDuJour; %>
<% dateDuJour = new Date();%>
Date du jour : <%= dateDuJour %><BR>
</body>
</html>
```

Par défaut, le code inclus dans le tag est inséré dans la méthode service() de la servlet générée à partir de la JSP.

Ce tag ne peut pas contenir autre chose que du code Java : il ne peut pas par exemple contenir de tags HTML ou JSP. Pour faire cela, il faut fermer le tag du scriptlet, mettre le tag HTML ou JSP puis de nouveau commencer un tag de scriptlet pour continuer le code.

Exemple :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<% for (int i=0; i<10; i++) { %>
<%= i %> <br>
<% }%>
</BODY>
</HTML>
```

Résultat : la page HTML générée

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
0 <br>
1 <br>
2 <br>
3 <br>
4 <br>
5 <br>
6 <br>
7 <br>
8 <br>
9 <br>
</BODY>
</HTML>
```

36.4.3. Les tags de commentaires

Il existe deux types de commentaires avec les JSP :

- les commentaires visibles dans le code HTML
- les commentaires invisibles dans le code HTML

36.4.3.1. Les commentaires HTML <!−− ... −−>

Ces commentaires sont ceux définis par format HTML. Ils sont intégralement reconduits dans le fichier HTML généré. Il est possible d'insérer, dans ce tag, un tag JSP de type expression qui sera exécuté.

La syntaxe est la suivante :

<!−− commentaires [<%= expression %>] −−>

Exemple :

```
<%@ page import="java.util.*" %>
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<!-- Cette page a ete generee le <%= new Date() %> -->
<p>Bonjour</p>
</BODY>
</HTML>
```

Résultat :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<!-- Cette page a ete generee le Thu Feb 15 11:44:25 CET 2001 -->
<p>Bonjour</p>
</BODY>
</HTML>
```

Le contenu d'une expression incluse dans des commentaires est dynamique : sa valeur peut changer à chaque génération de la page en fonction de son contenu.

36.4.3.2. Les commentaires cachés <%-- ... --%>

Les commentaires cachés sont utilisés pour documenter la page JSP. Leur contenu est ignoré par le moteur de JSP et ne sont donc pas reconduits dans la page HTML générée.

La syntaxe est la suivante :

<%-- commentaires --%>

Exemple :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<%-- Commentaires de la page JSP --%>
<p>Bonjour</p>
</BODY>
</HTML>
```

Résultat :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<p>Bonjour</p>
</BODY>
</HTML>
```

Ce tag peut être utile pour éviter l'exécution de code lors de la phase de débogage.

36.4.4. Les tags d'actions

Les tags d'actions permettent de réaliser des traitements couramment utilisés.

36.4.4.1. Le tag <jsp:useBean>

Le tag <jsp:useBean> permet de localiser une instance ou d'instancier un bean pour l'utiliser dans la JSP.

L'utilisation d'un bean dans une JSP est très pratique car il peut encapsuler des traitements complexes et être réutilisable par d'autre JSP ou composants. Le bean peut par exemple assurer l'accès à une base de données. L'utilisation des beans permet de simplifier les traitements inclus dans la JSP.

Lors de l'instanciation d'un bean, on précise la porté du bean. Si le bean demandé est déjà instancié pour la portée précisée alors il n'y pas de nouvelle instance du bean qui est créée mais sa référence est simplement renvoyée : le tag <jsp:useBean> n'instancie donc pas obligatoirement un objet.

Ce tag ne permet pas de traiter directement des EJB.

La syntaxe est la suivante :

<jsp:useBean
id="*beanInstanceNam*e"
scope="**page**|request|session|application"
{ class="*package.class*" |
type="*package.class*" |
class="*package.class*" type="*package.class*" |
beanName="{*package.class* | <%= *expression* %>}" type="*package.class*"
}
{ /> |
> ...
</jsp:useBean>
}

L'attribut id permet de donner un nom à la variable qui va contenir la référence sur le bean.

L'attribut scope permet de définir la portée durant laquelle le bean est défini et utilisable. La valeur de cette attribut détermine la manière dont le tag localise ou instancie le bean. Les valeurs possibles sont :

Valeur	Rôle
page	Le bean est utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus. C'est la valeur par défaut.
request	le bean est accessible durant la durée de vie de la requête. La méthode getAttribute() de l'objet request permet d'obtenir une référence sur le bean.
session	le bean est utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent. La JSP qui créé le bean doit avoir l'attribut session = « true » dans sa directive page.
application	le bean est utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean. Le bean n'est instancié que lors du rechargement de l'application.

L'attribut class permet d'indiquer la classe du bean.

L'attribut type permet de préciser le type de la variable qui va contenir la référence du bean. La valeur indiquée doit obligatoirement être une super classe du bean ou une interface implémentée par le bean (directement ou par héritage)

L'attribut beanName permet d'instancier le bean grâce à la méthode instanciate() de la classe Beans.

Exemple :

```
<jsp:useBean id="monBean" scope="session" class="test.MonBean" />
```

Dans cet exemple, une instance de MonBean est créée une seule et unique fois lors de la session. Dans la même session, l'appel du tag <jsp:useBean> avec le même bean et la même portée ne feront que renvoyer l'instance créée. Le bean est ainsi accessible durant toute la session.

Le tag <jsp:useBean> recherche si une instance du bean existe avec le nom et la portée précisée. Si elle n'existe pas, alors

une instance est créée. Si il y a instanciation du bean, alors les tags <jsp:setProperty> inclus dans le tag sont utilisés pour initialiser les propriétés du bean sinon ils sont ignorés. Les tags inclus entre les tags <jsp:useBean> et </jsp:useBean> ne sont exécutés que si le bean est instancié.

Exemple :

```
<jsp:useBean id="monBean" scope="session" class="test.MonBean" >
<jsp:setProperty name="monBean" property="*" />
</jsp:useBean>
```

Cet exemple a le même effet que le précédent avec une initialisation des propriétés du bean lors de son instanciation avec les valeurs des paramètres correspondants.

Exemple complet :

TestBean.jsp

```
<html>
<HEAD>
<TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
</HEAD>
<body>
<p>Test d'utilisation d'un Bean dans une JSP </p>
<jsp:useBean id="personne" scope="request" class="test.Personne" />
<p>nom initial = <%=personne.getNom() %></p>
<%
personne.setNom("mon nom");
%>
<p>nom mise à jour = <%= personne.getNom() %></p>
</body>
</html>
```

Personne.java

```
package test;
public class Personne {
  private String nom;
  private String prenom;

  public Personne() {
    this.nom = "nom par defaut";
    this.prenom = "prenom par defaut";
  }

  public void setNom (String nom) {
    this.nom = nom;
  }

  public String getNom() {
    return (this.nom);
  }

  public void setPrenom (String prenom) {
    this.prenom = prenom;
  }

  public String getPrenom () {
    return (this.prenom);
  }
}
```

Selon le moteur de JSP utilisé, les fichiers du bean doivent être placés dans un répertoire particulier pour être accessibles par la JSP.

Pour tester cette JSP avec Tomcat, il faut compiler le bean Personne dans le répertoire c:\jakarta−tomcat\webapps\examples\web−inf\classes\test et placer le fichier TestBean.jsp dans le répertoire c:\jakarta−tomcat\webapps\examples\jsp\test.

36.4.4.2. Le tag <jsp:setProperty >

Le tag <jsp:setProperty> permet de mettre à jour la valeur d'un ou plusieurs attributs d'un Bean. Le tag utilise le setter (méthode setXXX() ou XXX est le nom de la propriété avec la première lettre en majuscule) pour mettre à jour la valeur. Le bean doit exister grâce à un appel au tag <jsp:useBean>.

Il existe trois façons de mettre à jour les propriétés soit à partir des paramètres de la requête soit avec une valeur :

- alimenter automatiquement toutes les propriétés avec les paramètres correspondants de la requête
- alimenter automatiquement une propriété avec le paramètre de la requête correspondant
- alimenter une propriété avec la valeur précisée

La syntaxe est la suivante :

<jsp:setProperty name="*beanInstanceName*"
{ property="*" |
property="*propertyName*" [param=" *parameterName*"] |
property="*propertyName*" value="{*string* | <%= *expression*%>}"
}
/>

L'attribut name doit contenir le nom de la variable qui contient la référence du bean. Cette valeur doit être identique à celle de l'attribut id du tag <jsp:useBean> utilisé pour instancier le bean.

L'attribut property= «*» permet d'alimenter automatiquement les propriétés du bean avec les paramètres correspondants contenus dans la requête. Le nom des propriétés et le nom des paramètres doivent être identiques.

Comme les paramètres de la requête sont toujours fournis sous forme de String, une conversion est réalisée en utilisant la méthode valueOf() du wrapper du type de la propriété.

Exemple :
`<jsp:setProperty name="monBean" property="*" />`

L'attribut property="*propertyNam*e" [param="*parameterNam*e"] permet de mettre à jour un attribut du bean. Par défaut, l'alimentation est faite automatiquement avec le paramètre correspondant dans la requête. Si le nom de la propriété et du paramètre sont différents, il faut préciser l'attribut property et l'attribut param qui doit contenir le nom du paramètre qui va alimenter la propriété du bean.

Exemple :
`<jsp:setProperty name="monBean" property="nom"/>`

L'attribut property="*propertyNam*e" value="{*string* | <%= *expression* %>}" permet d'alimenter la propriété du bean avec une valeur particulière.

Exemple :

```
<jsp:setProperty name="monBean" property="nom" value="toto" />
```

Il n'est pas possible d'utiliser param et value dans le même tag.

Exemple : Cette exemple est identique au précédent

```
<html>
<HEAD>
<TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
</HEAD>
<body>
<p>Test d'utilisation d'un Bean dans une JSP </p>
<jsp:useBean id="personne" scope="request" class="test.Personne" />
<p>nom initial = <%= personne.getNom() %></p>
<jsp:setProperty name="personne" property="nom" value="mon nom" />
<p>nom mis à jour = <%= personne.getNom() %></p>
</body>
</html>
```

Ce tag peut être utilisé entre les tags <jsp:useBean> et </jsp:useBean> pour initialiser les propriétés du bean lors de son instanciation.

36.4.4.3. Le tag <jsp:getProperty>

Le tag <jsp:getProperty> permet d'obtenir la valeur d'un attribut d'un Bean. Le tag utilise le getter (méthode getXXX() ou XXX est le nom de la propriété avec la première lettre en majuscule) pour obtenir la valeur et l'insérer dans la page HTML généré. Le bean doit exister grâce à un appel au tag <jsp:useBean>.

La syntaxe est la suivante :

<jsp:getProperty name="*beanInstanceName*" property=" *propertyName*" />

L'attribut name indique le nom du bean tel qu'il a été déclaré dans le tag <jsp:useBean>.

L'attribut property indique le nom de la propriété dont on veut la valeur.

Exemple :

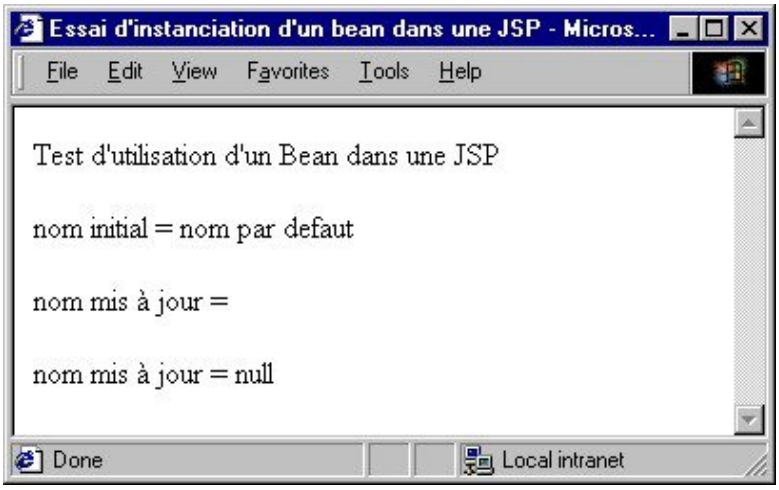
```
<html>
<HEAD>
<TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
</HEAD>
<body>
<p>Test d'utilisation d'un Bean dans une JSP </p>
<jsp:useBean id="personne" scope="request" class="test.Personne" />
<p>nom initial = <jsp:getProperty name="personne" property="nom" /></p>
<jsp:setProperty name="personne" property="nom" value="mon nom" />
<p>nom mise à jour = <jsp:getProperty name="personne" property="nom" /></p>
</body>
</html>
```

Attention : ce tag ne permet pas d'obtenir la valeur d'une propriété indexée ni les valeurs d'un attribut d'un EJB.

Remarque : avec Tomcat 3.1, l'utilisation du tag <jsp:getProperty> sur un attribut dont la valeur est null n'affiche rien alors que l'utilisation d'un tag d'expression retourne « null ».

Exemple :

```
<html>
<HEAD>
<TITLE>Essai d'instanciation d'un bean dans une JSP</TITLE>
</HEAD>
<body>
<p>Test d'utilisation d'un Bean dans une JSP </p>
<jsp:useBean id="personne" scope="request" class="test.Personne" />
<p>nom initial = <jsp:getProperty name="personne" property="nom" /></p>
<% personne.setNom(null);%>
<p>nom mis à jour = <jsp:getProperty name="personne" property="nom" /></p>
<p>nom mis à jour = <%= personne.getNom() %></p>
</body>
</html>
```



36.4.4.4. Le tag de redirection <jsp:forward>

Le tag <jsp:forward> permet de rediriger la requête vers une autre URL pointant vers un fichier HTML, JSP ou un servlet.

Dès que le moteur de JSP rencontre ce tag, il redirige le requête vers l'URL précisée et ignore le reste de la JSP courante. Tout ce qui a été généré par la JSP est perdu.

La syntaxe est la suivante :

<jsp:forward page="{*relativeURL* | <%= *expression* %>}" />
ou
<jsp:forward page="{*relativeURL* | <%= *expression* %>}" >
<jsp:param name="*parameterName*" value="{ *parameterValue* | <%= *expression* %>}" /> +
</jsp:forward>

L'option page doit contenir la valeur de l'URL de la ressource vers laquelle la requête va être redirigée.

Cette URL est absolue si elle commence par un '/' sinon elle est relative à la JSP . Dans le cas d'une URL absolue, c'est le serveur web qui détermine la localisation de la ressource.

Il est possible de passer un ou plusieurs paramètres vers la ressource appelée grâce au tag <jsp :param>.

Exemple :

Test8.jsp

```
<html>
<body>
<p>Page initiale appelée</p>
<jsp:forward page="forward.htm"/>
</body>
```

```
</html>
forward.htm
<HTML>
<HEAD>
<TITLE>Page HTML</TITLE>
</HEAD>
<BODY>
<p><table border="1" cellpadding="4" cellspacing="0" width="30%" align=center >
<tr bgcolor="#A6A5C2">
<td align="center">Page HTML forwardée</Td>
</Tr>
</table></p>
</BODY>
</HTML>
```

Dans l'exemple, le fichier forward.htm doit être dans le même répertoire que la JSP. Lors de l'appel à la JSP, c'est le page HTML qui est affichée. Le contenu généré par la page JSP n'est pas affiché.

36.4.4.5. Le tag <jsp:include>

Ce tag permet d'inclure le contenu généré par une JSP ou une servlet dynamiquement au moment ou la JSP est executée. C'est la différence avec la directive include avec laquelle le fichier est inséré dans la JSP avant la génération de la servlet.

La syntaxe est la suivante :

<jsp:include page="*relativeURL" flush="true"* />

L'attribut page permet de préciser l'URL relative de l'élément à insérer.

L'attribut flush permet d'indiquer si le tampon doit être envoyé au client et vidé. Si la valeur de ce paramètre est true, il n'est pas possible d'utiliser certaines fonctionnalités dans la servlet ou la JSP appelée : il n'est pas possible de modifier l'entete de la réponse (header, cookies) ou renvoyer ou faire suivre vers une autre page.

Exemple :

```
<html>
  <body>
    <jsp:include page="bandeau.jsp"/>
    <H1>Bonjour</H1>
    <jsp:include page="pied.jsp"/>
  </body>
</html>
```

Il est possible de fournir des paramètres à la servlet ou à la JSP appelée en utilisant le tag <jsp:param>.

36.4.4.6. Le tag <jsp:plugin>

Ce tag permet la génération du code HTML nécessaire à l'exécution d'une applet en fonction du navigateur : un tag HTML <Object> ou <Embed> est généré en fonction de l'attribut User−Agent de la requête.

Le tag <jsp:plugin> possède trois attributs obligatoires :

Attribut	Rôle
code	permet de préciser le nom de classe
codebase	contient une URL précisant le chemin absolu ou relatif du répertoire contenant la classe ou l'archive
type	les valeurs possibles sont applet ou bean

Il possède aussi plusieurs autres attributs optionnels dont les plus utilisés sont :

Attribut	Rôle
align	permet de préciser l'alignement de l'applet : les valeurs possibles sont bottom, middle ou top
archive	permet de préciser un ensemble de ressources (bibliothèques jar, classes, ...) qui seront automatiquement chargées. Le chemin de ces ressources tient compte de l'attribut codebase
height	précise la hauteur de l'applet en pixel ou en pourcentage
hspace	précise le nombre de pixels insérés à gauche et à droite de l'applet
jrversion	précise la version minimale du jre à utiliser pour faire focntionner l'applet
name	précise le nom de l'applet
vspace	précise le nombre de pixels insérés en haut et en bas de l'applet
width	précise la longueur de l'applet en pixel ou en pourcentage

Pour fournir un ou plusieurs paramètres, il faut utiliser dans le corps du tag <jsp:plugin> le tag <jsp:params>. Chaque paramètre sera alors défini dans un tag <jsp:param>.

Exemple :

```
<jsp:plugin type="applet" code="MonApplet.class" codebase="applets"
          jreversion="1.1" width="200" height="200" >
  <jsp:params>
     <jsp:param name="couleur" value="eeeeee" />
  </jsp:params>
</jsp:plugin>
```

Le tag <jsp:fallback> dans le corps du tag <jsp:plugin> permet de préciser un message qui sera affiché dans les navigateurs ne supportant pas le tag HTML <Object> ou <Embed>.

36.5. Un Exemple très simple

Exemple :

```
TestJSPIdent.html

<HTML>
<HEAD>
<TITLE>Identification</TITLE>
</HEAD>
<BODY>
<FORM METHOD=POST ACTION="jsp/TestJSPAccueil.jsp">
Entrer votre nom :
<INPUT TYPE=TEXT NAME="nom">
<INPUT TYPE=SUBMIT VALUE="SUBMIT">
</FORM>
</BODY>
</HTML>

TestJSPAccueil.jsp

<HTML>
<HEAD>
<TITLE>Accueil</TITLE>
</HEAD>
<BODY>
<%
```

```
String nom = request.getParameter("nom");
%>
<H2>Bonjour <%= nom %></H2>
</BODY>
</HTML>
```

36.6. La gestion des erreurs

Lors de l'exécution d'une page JSP, des erreurs peuvent survenir. Chaque erreur se traduit par la levée d'une exception. Si cette exception est capturée dans un bloc try/catch de la JSP, celle ci est traitée. Si l'exception n'est pas capturée dans la page, il y a deux possibilités selon qu'une page d'erreur soit associée à la page JSP :

- sans page d'erreur associée, la pile d'exécution de l'exception est affichée
- avec une page d'erreur associée, une redirection est effectuée vers cette JSP

La définition d'une page d'erreur permet de la préciser dans l'attribut errorPage de la directive page des autres JSP de l'application. Si une exception est levée dans les traitements d'une de ces pages, la JSP va automatiquement rediriger l'utilisateur va la page d'erreur précisée.

La valeur de l'attribut errorPage de la directive page doit contenir l'URL de la page d'erreur. Le plus simple est de définir cette page à la racine de l'application web et de faire précéder le nom de la page par un caractère '/' dans l'url.

Exemple :

```
<%@ page errorPage="/mapagederreur.jsp" %>
```

36.6.1. La définition d'une page d'erreur

Une page d'erreur est une JSP dont l'attribut isErrorPage est égal à true dans la directive page. Une telle page dispose d'un accès à la variable implicite nommée exception de type Throwable qui encapsule l'exception qui a été levée.

Il est possible dans une telle page d'afficher un message d'erreur personnalisé mais aussi d'inclure des traitements liés à la gestion de l'exception : ajouter l'exception dans un journal, envoie d'un mail pour son traitement, ...

Exemple :

```
<%@ page language="java" contentType="text/html" %>
%@ page isErrorPage="true" %>
<html>
  <body>
    <h1>Une erreur est survenue lors des traitements</h1>
    <p><%= exception.getMessage() %></p>
</body>
</html>
```

36.7. Les bibliothèques de tag personnalisées (custom taglibs)

Les bibliothèques de tags (taglibs) ou tags personnalisés (custom tags) permettent de définir ses propres tags basés sur XML, de les regrouper dans une bibliothèque et de les réutiliser dans des JSP. C'est une extension de la technologie JSP apparue à partir de la version 1.1 des spécifications des JSP.

36.7.1. Présentation

Un tag personnalisé est un élément du langage JSP défini par un développeur pour des besoins particuliers qui ne sont pas traités en standard par les JSP. Elles permettent de définir ces propres tags qui réaliseront des actions pour générer la réponse.

Le principal but est de favoriser la séparation des rôles entre le développeur Java et concepteur de page web. L'idée maitresse est de déporter le code Java contenu dans les scriplets de la JSP dans des classes dédiées et de les appeler dans le code source de la JSP en utilisant des tags particuliers.

Ce concept peut sembler proche de celui des javabeans dont le rôle principal est aussi de définir des composants réutilisables. Les javabeans sont particulièrements adaptés pour stocker et échanger des données entre les composants de l'application web via la session.

Les tags personnalisés sont adaptés pour enlever du code Java inclus dans les JSP est le déporter dans une classe dédiée. Cette classe est physiquement un javabean qui implémente une interface particulière.

La principale différence entre un javabean et un tag personnalisé est que ce dernier tient compte de l'environnement dans lequel il s'exécute (notamment la JSP et le contexte de l'application web) et interagit avec lui.

Pour de plus amples informations sur les bibliothèques de tags personnalisés, il suffit de consulter le site de Sun qui leur sont consacrées : http://java.sun.com/products/jsp/taglibraries.html.

Les tags personnalisés possèdent des fonctionnalités intéressantes :

- ils ont un accès aux objets de la JSP notamment l'objet de type HttpResponse. Ils peuvent donc modifier le contenu de la réponse générée par la JSP
- ils peuvent recevoir des paramètres envoyés à partir de la JSP qui les appelle
- ils peuvent avoir un corps qu'ils peuvent manipuler. Par extension de cette fonctionnalité, il est possible d'imbriquer un tag personnalisé dans un autre avec un nombre d'imbrication illimité

Les avantages des bibliothèques de tags personnalisés sont :

- un suppression du code Java dans la JSP remplacé par un tag XML facilement compréhensible ce qui simplifie grandement la JSP
- une API facile à mettre en oeuvre
- une forte et facile réutilisabilité des tags développés
- une maintenance des JSP facilitée

La définition d'une bibliothèque de tags comprend plusieurs entités :

- une classe dit "handler" pour chaque tag qui compose la bibliothèque
- un fichier de description de la bibliothèque

36.7.2. Les handlers de tags

Chaque tag est associé à une classe qui va contenir les traitements à exécuter lors de l'utilisation du tag. Une telle classe est nommée "handler de tag" (tag handler). Pour permettre leur appel, une telle classe doit obligatoirement implémenter directement ou indirectement l'interface javax.servlet.jsp.tagext.Tag

L'interface Tag possède une interface fille BodyTag qui doit être utilisée dans le cas ou le tag peut utiliser le contenu de son corps.

Pour plus de facilité, l'API JSP propose les classes TagSupport et BodyTagSupport qui implémentent respectivement l'interface Tag et BodyTag. Ces deux classes, contenues dans le package javax.servlet.jsp.tagext, proposent des implémentations par défaut des méthodes de l'interface. Ces deux classes proposent un traitement standard par défaut pour chacunes des méthodes de l'interface qu'ils implémentent. Pour définir un handler de tag, il suffit d'hériter de l'une ou l'autre de ces deux classes.

Les méthodes définies dans les interfaces Tag et BodyTag sont appelées, par la servlet issue de la compilation de la JSP, au cours de l'utilisation du tag.

Le cycle de vie général d'un tag est le suivant :

- lors de la rencontre du début du tag, un objet du type du handler est instancié
- plusieurs propriétés sont initialisées (pageContext, parent, ...) en utilisant les setters correspondant

- si le tag contient des attributs, les setters correspondant sont appelés pour alimenter leur valeur
- la méthode doStartTag() est appelée
- si la méthode doStartTag() renvoie la valeur EVAL_BODYINCLUDE alors le contenu du corps du tag est évalué
- lors de la rencontre de la fin du tag, appel de la méthode doEndTag()
- si la méthode doEndTag() renvoie la valeur EVAL_PAGE alors l'évaluation de la page se poursuit, si elle renvoie la valeur SKIP_PAGE elle ne se poursuit pas

Toutes ces opérations sont réalisées par le code généré lors de la compilation de la JSP.

Un handler de tag possède un objet qui permet d'avoir un accès aux objets implicites de la JSP. Cet objet est du type javax.servlet.jsp.PageContext

Comme le code contenu dans la classe du tag ne peut être utilisé que dans le contexte particulier du tag, il peut être intéressant de sortir une partie de ce code dans une ou plusieurs classes dédiées qui peuvent être éventuellement des beans.

Pour compiler ces classes, il faut obligatoirement que le jar de l'API servlets (servlets.jar) soit inclus dans la variable CLASSPATH.

36.7.3. L'interface Tag

Cette interface définit les méthodes principales pour la gestion du cycle de vie d'un tag personnalisé qui ne doit pas manipuler le contenu de son corps.

Elle définit plusieurs constantes :

Constante	Rôle
EVAL_BODY_INCLUDE	Continuer avec l'évaluation du corps du tag
EVAL_PAGE	Continuer l'évaluation de la page
SKIP_BODY	Empêcher l'évaluation du corps du tag
SKIP_PAGE	Empêcher l'évaluation du reste de la page

Elle définit aussi plusieurs méthodes :

Méthode	Rôle
int doEndTag()	Traitements à la rencontre du tag de début
int doStartTag()	Traitements à la rencontre du tag de fin
setPageContext(Context)	Sauvegarde du contexte de la page

La méthode doStartTag() est appelée lors de la rencontre du tag d'ouverture et contient les traitements à effectuer dans ce cas. Elle doit renvoyer un entier prédéfini qui indique comment va se poursuivre le traitement du tag :

- EVAL_BODY_INCLUDE : poursuite du traitement avec évaluation du corps du tag
- SKIP_BODY : poursuite du traitement sans évaluation du corps du tag

La méthode doEndTag() est appelée lors de la rencontre du tag de fermeture et contient les traitements à effectuer dans ce cas. Elle doit renvoyer un entier prédéfini qui indique comment va se poursuivre le traitement de la JSP.

- EVAL_PAGE : poursuite du traitement de la JSP
- SKIP_PAGE : ne pas poursuivre le traitement du reste de la JSP

36.7.4. L'accès aux variables implicites de la JSP

Les tags ont accès aux variables implicites de la JSP dans laquelle ils s'exécutent via un objet de type PageContext. La variable pageContext est un objet de ce type qui est initialisé juste après l'instanciation du handler.

Le classe PageContext est une classe abstraite dont l'implémentation des spécifications doit fournir une adaptation concrète.

Cette classe définit plusieurs méthodes :

Méthodes	Rôles
JspWriter getOut()	Permet un accès à la variable out de la JSP
Exception getException()	Permet un accès à la variable exception de la JSP
Object getPage()	Permet un accès à la variable page de la JSP
ServletRequest getRequest()	Permet un accès à la variable request de la JSP
ServletResponse getResponse()	Permet un accès à la variable response de la JSP
ServletConfig getServletConfig()	Permet un accès à l'instance de la variable de type ServletConfig
ServletContext getServletContext()	Permet un accès à l'instance de la variable de type ServletContext
HttpSession getSession()	Permet un accès à la session
Object getAttribute(String)	Renvoie l'objet associé au nom fourni en paramètre dans la portée de la page
setAttribute(String, Object)	Permet de placer dans la portée de la page un objet dont le nom est fourni en paramètre

36.7.5. Les deux types de handlers

Il existe deux types de handlers :

- les handlers de tags sans corps
- les handlers de tags avec corps

36.7.5.1. Les handlers de tags sans corps

Pour définir le handler d'un tag personnalisé sans corps, il suffit de définir une classe qui implémente l'interface Tag ou qui héritent de la classe TagSupport. Il faut définir ou redéfinir les méthodes doStartTag() et endStartTag()

La méthode doStartTag() est appelée à la rencontre du début du tag. Cette méthode doit contenir le code à exécuter dans ce cas et renvoyer la constante SKIP_BODY puisque le tag ne contient pas de corps

36.7.5.2. Les handlers de tags avec corps

Le cycle de vie d'un tel tag inclus le traitement du corps si la méthode doStartTag() renvoie la valeur EVAL_BODY_TAG.

Dans ce cas, les opérations suivantes sont réalisées :

- la méthode setBodyContent() est appelée
- le contenu du corps est traité
- la méthode doAfterBody() est appelée. Si elle renvoie la valeur EVAL_BODY_TAG, le contenu du corps est de nouveau traité

36.7.6. Les paramètres d'un tag

Un tag peut avoir un ou plusieurs paramètres qui seront transmis à la classe via des attributs. Pour chacun des paramètres, il faut définir des getter et des setter en respectant les règles et conventions des Java beans. Il est impératif de définir un champ, un setter et éventuellement un accesseur pour chaque attribut.

La JSP utilisera le setter pour fournir à l'objet la valeur de l'attribut.

Au moment de la génération de la servlet par le moteur de JSP, celui ci vérifie par introspection la présence d'un setter pour l'attribut concerné.

36.7.7. Définition du fichier de description de la bibliothèque de tags (TLD)

Le fichier de description de la bibliothèque de tags (tag library descriptor file) est un fichier au format XML qui décrit une bibliothèque de tag. Les informations qu'il contient concerne la bibliothèque de tags elle même et concerne aussi chacun des tags qui la compose.

Ce fichier est utilisé par le conteneur Web lors de la compilation de la JSP pour remplacer le tag par du code Java.

Ce fichier doit toujours avoir pour extension .tld. Il doit être placé dans le répertoire web−inf du fichier war ou dans un de ces sous répertoires. Le plus pratique est de tous les regrouper dans un répertoire nommé par exemple tags ou tld.

Comme tout bon fichier XML, le fichier TLD commence par un prologue :

Exemple :

```
<?xml version="1.0" encoding="ISO−8859−1" ?>
<!DOCTYPE taglib
  PUBLIC "−//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
  "http://java.sun.com/j2ee/dtds/web−jsptaglibrary_1_1.dtd">
```

La DTD précisée doit correspondre à la version de l'API JSP utilisée. L'exemple précédent concernait la version 1.1, l'exemple suivant concerne la version 1.2

Exemple :

```
<?xml version="1.0" encoding="ISO−8859−1" ?>
<!DOCTYPE taglib
  PUBLIC "−//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtds/web−jsptaglibrary_1_2.dtd">
```

Le tag racine du document XML est le tag <taglib>.

Ce tag peut contenir plusieurs tags qui définissent les caractéristiques générales de la bibliothèque. Les tags suivants sont définis dans les spécifications 1.2 :

Nom	Rôle
tlib−version	version de la bibliothèque
jsp−version	version des spécifications JSP utilisée

short−name	nom court la bibliothèque (optionnel)
uri	URI qui identifie de façon unique la bibliothèque : cette URI n'a pas besoin d'exister réellement
display−name	nom de la bibliothèque
small−icon	(optionnel)
large−icon	(optionnel)
description	description de la bibliothèque
validator	(optionnel)
listener	(optionnel)
tag	il en faut autant que de tags qui composent la bibliothèque

Pour chaque tag personnalisé défini dans la bibliothèque, il faut un tag <tag>. Ce tag permet de définir les caractérisques d'un tag de la bibliothèque.

Ce tag peut contenir les tags suivants :

Nom	Rôle
name	nom du tag : il doit être unique dans la bibliothèque
tag−class	nom entièrement qualifié de la classe qui contient le handler du tag
tei−class	nom qualifié d'une classe fille de la classe javax.servlet.jsp.tagext.TagExtraInfo (optionnel)
body−content	type du corps du tag. Les valeurs possibles sont : • JSP : le corps du tag contient des tags JSP qui doivent être interprétés • tagdependent : l'interprétation du contenu du corps est faite par le tag • empty : le corps doit obligatoirement être vide La valeur par défaut est JSP
display−name	nom court du tag
small−icon	nom relatif par rapport à la bibliothèque d'un fichier gif ou jpeg contenant une icône. (optionnel)
large−icon	nom relatif par rapport à la bibliothèque d'un fichier gif ou jpeg contenant une icône. (optionnel)
description	description du tag (optionnel)
variable	(optionnel)
attribute	il en faut autant que d'attribut possédé par le tag (optionnel)
example	un exemple de l'utilisation du tag (optionnel)

Pour chaque attribut du tag personnalisé, il faut utiliser un tag <attribute>. Ce tag décrit un attribut d'un tag et peut contenir les tags suivants :

Nom	Description
name	nom de l'attribut
required	booléen qui indique la présence obligatoire de l'attribut
rtexprvalue	booléen qui indique si la page doit évaluer l'expression lors de l'exécution. Il faut donc mettre la valeur true si la valeur de l'attribut est fournie avec un tag JSP d'expression <%= %>

Le tag <Variable> contient les tags suivants :

Nom	Rôle
name−given	
name−from−attribut	
variable−class	nom de la classe de la valeur de l'attribut. Par défaut java.lang.String
declare	par défaut : True
scope	visibilité de l'attribut. Les valeurs possibles sont : • AT_BEGIN • NESTED • AT_END Par défaut : NESTED (optionnel)
description	description de l'attribut (optionnel)

Chaque bibliothèque doit être définie avec un fichier de description au format xml possèdant une extension .tld. Le contenu de ce fichier doit pouvoir être validé avec une DTD fournie par Sun.

Ce fichier est habituellement stocké dans le répertoire web−inf de l'application web ou un de ses sous répertoires.

Exemple :

```
<?xml version="1.0" encoding="ISO−8859−1" ?>
  <!DOCTYPE taglib PUBLIC "−//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
  "http://java.sun.com/j2ee/dtds/web−jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>testtaglib</shortname>
  <uri>http://perso.jmd.test.taglib</uri>
  <info>Bibliotheque de test des taglibs</info>

  <tag>
  <name>testtaglib1</name>
  <tagclass>perso.jmd.test.taglib.TestTaglib1</tagclass>
  <info>Tag qui affiche bonjour</info>
  </tag>
</taglib>
```

36.7.8. Utilisation d'une bibliothèque de tags

Pour utiliser une bibliothèque de classe, il y a des actions à réaliser au niveau du code source de la JSP et au niveau de conteneur d'application web pour déployer la bibliothèque de tags.

36.7.8.1. Utilisation dans le code source d'une JSP

Pour chaque bibliothèque à utiliser dans une JSP, il faut la déclarer en utilisant la directive taglib avant son utilisation. Le plus simple est d'effectuer ces déclarations tout au début du code de la JSP.

Cette directive possède deux attributs :

- uri : l'URI de la bibliothèque telle que définie dans le fichier de description
- prefix : un préfix qui servira d'espace de noms pour les tags de la bibliothèque dans la JSP

Exemple :

```
<%@ taglib uri="/WEB-INF/tld/testtaglib.tld" prefix="maTagLib" %>
```

L'attribut uri permet de donner une identitée au fichier de description de la bibliothèque de tags (TLD). La valeur fournie peut être :

- directe (par exemple le nom du fichier avec son chemin relatif)

Exemple :

```
<%@ taglib uri="/WEB-INF/tld/testtaglib.tld" prefix="maTagLib" %>
```

- ou indirecte (concordance avec un nom logique défini dans un tag taglib du descripteur de déploiement de l'application web)

Exemple :

```
<%@ taglib uri= “/maTaglib” prefix= “maTagbib” %>
```

Dans ce dernier cas, il faut ajouter pour chaque bibliothèque un tag <taglib> dans le fichier de description de déploiement de l'application/WEB−INF/web.xml

Exemple :

```
 <taglib>
    <taglib-uri>/maTagLibTest</taglib-uri>
    <taglib-location>/WEB-INF/tld/testtaglib.tld</taglib-location>
  </taglib>
```

L'appel d'un tag se fait en utilisant un tag dont le nom à la forme suivante : prefix:tag

Le préfix est celui défini dans la directive taglib.

Exemple : un tag sans corps :

```
<maTagLib:testtaglib1/>
```

Exemple : une tag avec corps :

```
<prefix:tag>
   ...
</prefix:tag>
```

Le corps peut contenir du code HTML, du code JSP ou d'autre tag personnalisé.

Le tag peut avoir des attributs si ceux ci ont été définis. La syntaxe pour les utiliser respecte la norme XML

Exemple : un tag avec un paramètre constant :

```
<prefix:tag attribut="valeur"/>
```

La valeur de cet attribut peut être une donnée dynamiquement évaluée lors de l'exécution :

Exemple : un tag avec un paramètre :

```
<prefix:tag attribut="<%= uneVariable%>"/>
```

36.7.8.2. Déploiement d'une bibliothèque

Au moment de la compilation de la JSP en servlet, le conteneur transforme chaque tag en un appel à un objet du type de la classe associé au tag.

Il y a deux types d'éléments dont il faut s'assurer l'accès par le conteneur d'applications web :

- le fichier de description de la bibliothèque
- les classes des handlers de tag

Les classes des handlers de tags peuvent être stockées à deux endroits dans le fichier war selon leur format :

- si ils sont packagés sous forme de fichier jar alors ils doivent être placés dans le répertoire /WEB−INF/lib
- si ils ne sont pas packagés alors ils doivent être placés dans le répertoire /WEB−INF/classes

36.7.9. Déploiement et tests dans Tomcat

Tomcat étant l'implémentation de référence pour les technologies servlets et JSP, il est pratique d'effectuer des tests avec cet outil.

La version de Tomcat utilisée dans cette section est la 3.2.1.

36.7.9.1. Copie des fichiers

Les classes compilées doivent être copiées dans le répertoire WEB−INF/classes de la webapp si elles ne sont pas packagées dans une archive jar, sinon le ou les fichiers .jar doivent être copiés dans le répertoire WEB−INF/lib.

Le fichier .tld doit être copié dans le répertoire WEB−INF ou dans un de ces sous répertoires.

36.7.9.2. Enregistrement de la bibliothèque

Il faut enregistrer la bibliothèque dans le fichier de configuration web.xml contenu dans le répertoire web−inf du répertoire de l'application web.

Il faut ajouter dans ce fichier, un tag <taglib> pour chaque bibliothèque utilisée par l'application web contenant deux informations :

- l'URI de la bibliothèque contenue dans le tag taglib−uri. Cette URI doit être identique à celle définie dans le fichier de description de la bibliothèque
- la localisation du fichier de description

Exemple :

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<!DOCTYPE web−app
  PUBLIC "−//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web−app_2.2.dtd">
<web−app>
  <welcome−file−list id="ListePageDAccueil">
    <welcome−file>index.htm</welcome−file>
    <welcome−file>index.jsp</welcome−file>
  </welcome−file−list>
```

```
  <taglib>
    <taglib-uri>/maTagLibTest</taglib-uri>
    <taglib-location>/WEB-INF/tld/testtaglib.tld</taglib-location>
  </taglib>
</web-app>
```

36.7.9.3. Test

Il ne reste plus qu'à lancer Tomcat si ce n'est pas encore fait et de saisir l'url de la page contenant l'appel au tag personnalisé.

36.7.10. Les bibliothèques de tags existantes

Il existe de nombreuses bibliothèques de tags libres ou commerciales disponibles sur le marché. Cette section va tenter de présenter quelques unes de plus connues et des plus utilisées du monde libre. Cette liste n'est pas exhaustive.

36.7.10.1. Struts

Struts est un framework pour la réalisation d'applications web reposant sur le modèle MVC 2.

Pour la partie vue, Struts utilise les JSP et propose en plus plusieurs bibliothèques de tags pour faciliter le développement de cette partie présentation. Struts possède quatre grandes bibliothèques :

- formulaire HMTL
- modèles (templates)
- Javabeans (bean)
- traitements logiques (logic)

Le site web de struts se trouve à l'url : http://jakarta.apache.org/struts/index.html

Le chapitre "Les frameworks pour les applications web" fournit une section sur la mise en oeuvre et l'utilisation de Struts.

36.7.10.2. Jakarta Tag libs

La suite de ce chapitre sera développée dans une version future de ce document

36.7.10.3. JSP Standard Tag Library (JSTL)

JSP Standard Tag Library (JSTL) est une spécification issu du travail du JCP sous la JSR numéro 52. Le chapitre sur JSTL fournit plus de détails sur cette spécification.

37. JSTL (Java server page Standard Tag Library)

JSTL est l'acronyme de Java server page Standard Tag Library. C'est un ensemble de tags personnalisés développé sous la JSR 052 qui propose des fonctionnalités souvent rencontrées dans les JSP :

- Tag de structure (itération, conditionnement ...)
- Internationalisation
- Exécution de requête SQL
- Utilisation de document XML

JSTL nécessite un conteneur d'application web qui implémente l'API servlet 2.3 et l'API JSP 1.2. L'implémentation de référence (JSTL−RI) de cette spécification est développée par le projet Taglibs du groupe Apache sous le nom " Standard ".

Il est possible de télécharger cette implémentation de référence à l'URL :
http://jakarta.apache.org/taglibs/doc/standard−doc/intro.html

JSTL est aussi inclus dans le JWSDP (Java Web Services Developer Pack), ce qui facilite son installation et son utilisation. Les exemples de cette section ont été réalisés avec le JWSDP 1.001

JSTL possède quatre bibliothèques de tag :

Rôle	TLD	Uri
Fonctions de base	c.tld	http://java.sun.com/jstl/core
Traitements XML	x.tld	http://java.sun.com/jstl/xml
Internationalisation	fmt.tld	http://java.sun.com/jstl/fmt
Traitements SQL	sql.tld	http://java.sun.com/jstl/sql

JSTL propose un langage nommé EL (expression langage) qui permet de faire facilement référence à des objets java accessibles dans les différents contextes de la JSP.

La bibliothèque de tag JSTL est livrée en deux versions :

- JSTL−RT : les expressions pour désigner des variables utilisant la syntaxe JSP classique
- JSTL−EL : les expressions pour désigner des variables utilisant le langage EL

Pour plus informations, il est possible de consulter les spécifications à l'url suivante :
http://jcp.org/aboutJava/communityprocess/final/jsr052/

Ce chapitre contient plusieurs sections :

- La bibliothèque Database

37.1. Un exemple simple

Pour commencer, voici un exemple et sa mise en oeuvre détaillée. L'application web d'exemple se nomme test. Il faut créer un répertoire test dans le répertoire webapps de tomcat.

Pour utiliser JSTL, il faut copier les fichiers jstl.jar et standard.jar dans le répertoire WEB−INF/lib de l'application web.

Il faut copier les fichiers .tld dans le répertoire WEB−INF ou un de ses sous répertoires. Dans la suite de l'exemple, ces fichiers ont été placés le répertoire /WEB−INF/tld.

Il faut ensuite déclarer les bibliothèques à utiliser dans le fichier web.xml du répertoire WEB−INF comme pour toute bibliothèque de tags personnalisés.

Exemple : pour la bibliothèque Core :

```
<taglib>
    <taglib−uri>http://java.sun.com/jstl/core</taglib−uri>
<taglib−location>/WEB−INF/tld/c.tld</taglib−location>
</taglib>
```

L'arborescence des fichiers est la suivante :

```
webapps
      test
            WEB−INF
                  lib
                        jstl.jar
                        standard.jar
                  tld
                        c.tld
                  web.xml
            test.jsp
```

Pour pouvoir utiliser une bibliothèque personnalisée, il faut utiliser la directive taglib :

Exemple :

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

Voici les code sources des différents fichiers de l'application web :

Exemple : fichier test.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Exemple</title>
  </head>

  <body>
      <c:out value="Bonjour" /><br/>
  </body>
</html>
```

Exemple : le fichier WEB−INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app23.dtd">

<web-app>
  <taglib>
    <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
<taglib-location>/WEB-INF/tld/c.tld</taglib-location>
</taglib>
</web-app>
```

Pour tester l'application, il suffit de lancer Tomcat et de saisir l'url localhost:8080/test/test.jsp dans un browser.

37.2. Le langage EL (Expression Langage)

JSTL propose un langage particulier constitué d'expressions qui permet d'utiliser et de faire référence à des objets java accessible dans les différents contexte de la page JSP. Le but est de fournir un moyen simple d'accéder aux données nécessaires à une JSP.

La syntaxe de base est ${xxx} ou xxx est le nom d'une variable d'un objet java défini dans un contexte particulier. La définition dans un contexte permet de définir la portée de la variable (page, requete, session ou application).

EL permet facilement de s'affranchir de la syntaxe de java pour obtenir une variable.

Exemple : accéder à l'attribut nom d'un objet personne situé dans la session avec Java

```
<%= session.getAttribute("personne").getNom() %>
```

Exemple : accéder à l'attribut nom d'un objet personne situé dans la session avec EL

```
${sessionScope.personne.nom}
```

EL possède par défaut les variables suivantes :

Variable	Rôle
PageScope	variable contenue dans la portée de la page (PageContext)
RequestScope	variable contenue dans la portée de la requête (HttpServletRequest)
SessionScope	variable contenue dans la portée de la session (HttpSession)
ApplicationScope	variable contenue dans la portée de l'application (ServletContext)
Param	paramètre de la requête http
ParamValues	paramètres de la requête sous la forme d'une collection
Header	en tête de la requête
HeaderValues	en têtes de la requête sous la forme d'une collection
InitParam	paramètre d'initialisation
Cookie	cookie
PageContext	objet PageContext de la page

EL propose aussi différents opérateurs :

Operateur	Rôle	Exemple
.	Obtenir une propriété d'un objet	${param.nom}
[]	Obtenir une propriété par son nom ou son indice	${param[" nom "]} ${row[1]}
Empty	Teste si un objet est null ou vide si c'est une chaîne de caractère. Renvoie un booleen	${empty param.nom}
== eq	test l'égalité de deux objet	
!= ne	test l'inégalité de deux objet	
< lt	test strictement inférieur	
> gt	test strictement supérieur	
<= le	test inférieur ou égal	
>= ge	test supérieur ou égal	
+	Addition	
−	Soustraction	
*	Multiplication	
/ div	Division	
% mod	Modulo	
&& and		
|| or		
! not	Négation d'un valeur	

EL ne permet pas l'accès aux variables locales. Pour pouvoir accéder à de telles variables, il faut obligatoirement en créer une copie dans une des portées particulières : page, request, session ou application

Exemple :

```
<%
  int valeur = 101;
%>
  valeur = <c:out value="${valeur}" /><BR/>
```

Résultat :

```
valeur =
```

Exemple : avec la variable copiée dans le contexte de la page

```
<%
  int valeur = 101;
  pageContext.setAttribute("valeur", new Integer(valeur));
%>
  valeur = <c:out value="${valeur}" /><BR/>
```

Résultat :

```
valeur = 101
```

37.3. La bibliothèque Core

Elle propose les tags suivants répartis dans trois catégories :

Catégorie	Tag
Utilisation de EL	set out remove catch
Gestion du flux (condition et itération)	if choose forEach forTokens
Gestion des URL	import url redirect

Pour utiliser cette bibliothèque, il faut la déclarer dans le fichier web.xml du répertoire WEB−INF de l'application web.

Exemple :

```
 <taglib>
    <taglib−uri>http://java.sun.com/jstl/core</taglib−uri>
    <taglib−location>/WEB−INF/tld/c.tld</taglib−location>
</taglib>
```

Dans chaque JSP qui utilise un ou plusieurs tags de la bibliothèque, il faut la déclarer avec une directive taglib

Exemple :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

37.3.1. Le tag set

Le tag set permet de stocker une variable dans une portée particulière (page, requete, session ou application).

Il possède plusieurs attributs :

Attribut	Rôle

value	valeur à stocker
target	nom de la variable contenant un bean dont la propriété doit être modifiée
property	nom de la propriété à modifier
var	nom de la variable qui va stocker la valeur
scope	portée de la variable qui va stocker la valeur

Exemple :

```
    <c:set var="maVariable1" value="valeur1" scope="page" />
    <c:set var="maVariable2" value="valeur2" scope="request" />
    <c:set var="maVariable3" value="valeur3" scope="session" />
    <c:set var="maVariable4" value="valeur4" scope="application" />
```

La valeur peut être déterminée dynamiquement.

Exemple :

```
    <c:set var="maVariable" value="${param.id}" scope="page" />
```

L'attribut target avec l'attribut property permet de modifier la valeur d'une propriété (précisée avec l'attribut property) d'un objet (précisé avec l'attribut target).

La valeur de la variable peut être précisée dans le corps du tag plutot que d'utiliser l'attribut value.

Exemple :

```
<c:set var="maVariable" scope="page">
       Valeur de ma variable
</c:set>
```

37.3.2. Le tag out

Le tag out permet d'envoyer dans le flux de sortie de la JSP le résultat de l'évaluation de l'expression fournie dans le paramètre " value ". Ce tag est équivalent au tag d'expression <%= ... %> de JSP.

Il possède plusieurs attributs :

Attribut	Rôle
value	valeur à afficher (obligatoire)
default	définir une valeur par défaut si la valeur est null
escapeXml	booléen qui precise si les caractères particuliers (< > & ...) doivent être convertis en leur équivalent HTML (< > &amp ; ...)

Exemple :

```
    <c:out value='${pageScope.maVariable1}' />
    <c:out value='${requestScope.maVariable2}' />
    <c:out value='${sessionScope.maVariable 3}' />
    <c:out value='${applicationScope.maVariable 4}' />
```

Il n'est pas obligatoire de préciser la portée dans laquelle la variable est stockée : dans ce cas, la variable est recherchée prioritairement dans la page, la requete, la session et enfin l'application.

L'attribut default permet de définir une valeur par défaut si le résultat de l'évaluation de la valeur est null. Si la valeur est null et que l'attribut default n'est pas utilisé alors c'est une chaîne vide qui est envoyée dans le flux de sortie.

Exemple :

```
<c:out value="${personne.nom}" default="Inconnu" />
```

Le tag out est particulièrement utile pour générer le code dans un formulaire en remplaçant avantageusement les scriplets.

Exemple :

```
<input type="text" name="nom" value="<c:out value="${param.nom}"/>" />
```

37.3.3. Le tag remove

Le tag remove permet de supprimer une variable d'une portée particulière.

Il possède plusieurs attributs :

Attribut	Rôle
var	nom de la variable à supprimer (obligatoire)
scope	portée de la variable

Exemple :

```
<c:remove var="maVariable1" scope="page" />
<c:remove var="maVariable2" scope="request" />
<c:remove var="maVariable3" scope="session" />
<c:remove var="maVariable4" scope="application" />
```

37.3.4. Le tag catch

Ce tag permet de capturer des exceptions qui sont levées lors de l'exécution du code inclus dans son corps.

Il possède un attribut :

Attribut	Rôle
var	nom d'une variable qui va contenir des informations sur l'anomalie

Si l'attribut var n'est pas utilisé, alors toutes les exceptions levées lors de l'exécution du corps du tag sont ignorées.

Exemple : code non protégé

```
<c:set var="valeur" value="abc" />
<fmt:parseNumber var="valeurInt" value="${valeur}"/>
```

Résultat : une exception est lévée

```
javax.servlet.ServletException: In <parseNumber>, value attribute can not be parsed: "abc"
        at org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:471)
        at org.apache.jsp.test$jsp.jspService(test$jsp.java:1187)
        at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:107)
```

L'utilisation du tag catch peut empecher le plantage de l'application.

Exemple :

```
<c:set var="valeur" value="abc" />
<c:catch var="erreur">
   <fmt:parseNumber var="valeurInt" value="${valeur}"/>
</c:catch>
<c:if test="${not empty erreur}">
   la valeur n'est pas numerique
</c:if>
```

Résultat :

```
la valeur n'est pas numerique
```

L'objet désigné par l'attribut var du tag catch possède une propriété message qui contient le message d'erreur

Exemple :

```
<c:set var="valeur" value="abc" />
<c:catch var="erreur">
   <fmt:parseNumber var="valeurInt" value="${valeur}"/>
</c:catch>
<c:if test="${not empty erreur}">
 <c:out value="${erreur.message}"/>
</c:if>
```

Résultat :

```
In &lt;parseNumber&gt;, value attribute can not be parsed: "abc"
```

Le souci avec ce tag est qu'il n'est pas possible de savoir quelle exception a été levée.

37.3.5. Le tag if

Ce tag permet d'évaluer le contenu de son corps si la condition qui lui est fournie est vraie.

Il possède plusieurs attributs :

Attribut	Rôle
test	condition à évaluer
var	nom de la variable qui contiendra le résultat de l'évaluation
scope	portée de la variable qui contiendra le résultat

Exemple :

```
<c:if test="${empty personne.nom}" >Inconnu</c:if>
```

Le tag peut ne pas avoir de corps si le tag est simplement utilisé pour stocker le résultat de l'évaluation de la condition dans une variable.

Exemple :

```
<c:if test="${empty personne.nom}" var="resultat" />
```

Le tag if est particulièrement utile pour générer le code dans un formulaire en remplaçant avantageusement les scriplets.

Exemple : selection de la bonne occurrence dont la valeur est fournie en paramètre de la requête

```
<FORM NAME="form1" METHOD="post" ACTION="">
  <SELECT NAME="select">
    <OPTION VALUE="choix1" <c:if test="${param.select == 'choix1'}" >selected</c:if> >
      choix 1</OPTION>
    <OPTION VALUE="choix2" <c:if test="${param.select == 'choix2'}" >selected</c:if> >
      choix 2</OPTION>
    <OPTION VALUE="choix3" <c:if test="${param.select == 'choix3'}" >selected</c:if> >
      choix 3</OPTION>
  </SELECT>
</FORM>
```

Pour tester le code, il faut fournir en paramètre dans l'url select=choix2

Exemple :

```
http://localhost:8080/test/test.jsp?select=choix2
```

37.3.6. Le tag choose

Ce tag permet de traiter différents cas mutuellement exclusifs dans un même tag. Le tag choose ne possède pas d'attribut. Il doit cependant posséder un ou plusieurs tags fils « when ».

Le tag when possède l'attribut test qui permet de préciser la condition à évaluer. Si la condition est vraie alors le corps du tag when est évaluer et le résultat est envoyé dans le flux de sortie de la JSP

Le tag otherwise permet de définir un cas qui ne correspond à aucun des autres inclus dans le tag. Ce tag ne possède aucun attribut.

Exemple :

```
<c:choose>
  <c:when test="${personne.civilite == 'Mr'}">
    Bonjour Monsieur
  </c:when>
  <c:when test="${personne.civilite == 'Mme'}">
    Bonjour Madame
  </c:when>
  <c:when test="${personne.civilite == 'Mlle'}">
    Bonjour Mademoiselle
  </c:when>
  <c:otherwise>
    Bonjour
  </c:otherwise>
</c:choose>
```

37.3.7. Le tag forEach

Ce tag permet de parcourir les différents éléments d'une collection et ainsi d'exécuter de façon répétitive le contenu de son corps.

Il possède plusieurs attributs :

Attribut	Rôle
var	nom de la variable qui contient l'élément en cours de traitement
items	collection à traiter
varStatus	nom d'un variable qui va contenir des informations sur l'itération en cours de traitement
begin	numéro du premier élément à traiter (le premier possède le numéro 0)
end	numéro du dernier élément à traiter
step	pas des éléments à traiter (par défaut 1)

A chaque itération, la valeur de la variable dont le nom est précisé par la propriété var change pour contenir l'élement de la collection en cours de traitement.

Aucun des attributs n'est obligatoire mais il faut obligatoirement qu'il y ait l'attribut items ou les attributs begin et end.

Le tag forEach peut aussi réaliser des itérations sur les nombres et non sur des éléments d'une collection. Dans ce cas, il ne faut pas utiliser l'attribut items mais uniquement utiliser les attributs begin et end pour fournir les bornes inférieures et supérieures de l'itération.

Exemple :

```
<c:forEach begin="1" end="4" var="i">
<c:out value="${i}"/><br>
</c:forEach>
```

Résultat :

```
1
2
3
4
```

L'attribut step permet de préciser le pas de l'itération.

Exemple :

```
<c:forEach begin="1" end="12" var="i" step="3">
  <c:out value="${i}"/><br>
</c:forEach>
```

Exemple :

```
1
4
7
10
```

L'attribut varStatus permet de définir une variable qui va contenir des informations sur l'itération en cours d'exécution. Cette variable possède plusieurs propriétés :

Attribut	Rôle
index	indique le numéro de l'occurrence dans l'ensemble de la collection
count	indique le numéro de l'itération en cours (en commençant par 1)
first	booléen qui indique si c'est la première itération
last	booléen qui indique si c'est la dernière itération

Exemple :

```
<c:forEach begin="1" end="12" var="i" step="3" varStatus="vs">
  index = <c:out value="${vs.index}"/> :
  count = <c:out value="${vs.count}"/> :
  value = <c:out value="${i}"/>
  <c:if test="${vs.first}">
     : Premier element
  </c:if>
  <c:if test="${vs.last}">
     : Dernier element
  </c:if>
  <br>
</c:forEach>
```

Résultat :

```
  index = 1 : count = 1 : value = 1 : Premier element
  index = 4 : count = 2 : value = 4
  index = 7 : count = 3 : value = 7
  index = 10 : count = 4 : value = 10 : Dernier element
```

37.3.8. Le tag forTokens

Ce tag permet de découper une chaîne selon un ou plusieurs séparateurs donnés et ainsi d'exécuter de façon répétitive le contenu de son corps autant de fois que d'occurrences trouvées.

Il possède plusieurs attributs :

Attribut	Rôle
var	variable qui contient l'occurrence en cours de traitement (obligatoire)
items	la chaîne de caractères à traiter (obligatoire)
delims	précise le séparateur
varStatus	nom d'un variable qui va contenir des informations sur l'itération en cours de traitement
begin	numero du premier élément à traiter (le premier possède le numéro 0)
end	numéro du dernier élément à traiter
step	pas des éléments à traiter (par défaut 1)

L'attribut delims peut avoir comme valeur une chaîne de caractères ne contenant qu'un seul caractère (délimiteur unique) ou un ensemble de caractères (délimiteurs multiples).

Exemple :

```
<c:forTokens var="token" items="chaine 1;chaine 2;chaine 3" delims=";">
  <c:out value="${token}" /><br>
</c:forTokens>
```

Exemple :

```
chaine 1
chaine 2
chaine 3
```

Dans le cas ou il y a plusieurs délimiteurs, chacun peut servir de séparateur

Exemple :

```
<c:forTokens var="token" items="chaine 1;chaine 2,chaine 3" delims=";,">
  <c:out value="${token}" /><br>
</c:forTokens>
```

Attention : Il n'y a pas d'occurrence vide. Dans le cas ou deux séparateurs se suivent consécutivement dans la chaîne à traiter, ceux ci sont considérés comme un seul séparateur. Si la chaîne commence ou se termine par un séparateur, ceux ci sont ignorés.

Exemple :

```
<c:forTokens var="token" items="chaine 1;;chaine 2;;;chaine 3" delims=";">
  <c:out value="${token}" /><br>
</c:forTokens>
```

Résultat :

```
chaine 1
chaine 2
chaine 3
```

Il est possible de ne traiter qu'un sous ensemble des occurrences de la collection. JSTL attribut à chaque occurrence un numéro incrémenter de 1 en 1 à partir de 0. Les attributs begin et end permettent de préciser une plage d'occurrence à traiter.

Exemple :

```
<c:forTokens var="token" items="chaine 1;chaine 2;chaine 3" delims=";" begin="1" end="1" >
  <c:out value="${token}" /><br>
</c:forTokens>
```

Résultat :

```
chaine 2
```

Il est possible de n'utiliser que l'attribut begin ou l'attribut end. Si seul l'attribut begin est précisé alors les n dernières occurrences seront traitées. Si seul l'attribut end est précisé alors seuls les n premières occurrences seront traitées.

Les attributs varStatus et step ont le même rôle que ceux du tag forEach.

37.3.9. Le tag import

Ce tag permet d'accéder à une ressource via son URL pour l'inclure ou l'utiliser dans les traitements de la JSP. La ressource accédée peut être dans une autre application.

Son grand intérêt par rapport au tag <jsp :include> est de ne pas être limité au contexte de l'application web.

Il possède plusieurs attributs :

Attribut	Rôle
url	url de la ressource (obligatoire)
var	nom de la variable qui va stocker le contenu de la ressource sous la forme d'une chaîne de caractère
scope	portée de la variable qui va stocker le contenu de la ressource
context	contexte de l'application web qui contient la ressource (si la ressource n'est pas l'application web courante)
charEncoding	jeu de caractères utilisé par la ressource
varReader	nom de la variable qui va stocker le contenu de la ressource sous la forme d'un objet de type java.io.Reader

L'attribut url permet de préciser l'url de la ressource. Cette url peut être relative (par rapport à l'application web) ou absolue.

Exemple :

```
<c:import url="/message.txt" /><br>
```

Par défaut, le contenu de la ressource est inclus dans la JSP. Il est possible de stocker le contenu de la ressource dans une chaîne de caractères en utilisant l'attribut var. Cet attribut attend comme valeur le nom de la variable.

Exemple :

```
<c:import url="/message.txt" var="message" />
<c:out value="${message}" /><BR/>
```

37.3.10. Le tag redirect

Ce tag permet de faire une redirection vers une nouvelle URL.

Les paramètres peuvent être fournis grace à un ou plusieurs tags fils param.

Exemple :

```
<c:redirect url="liste.jsp">
  <c:param name="id" value="123"/>
</c:redirect>
```

37.3.11. Le tag url

Ce tag permet de formatter une url. Il possède plusieurs attributs :

Attribut	Rôle
value	base de l'url (obligatoire)
var	nom de la variable qui va stocker l'url
scope	portée de la variable qui va stocker l'url
context	

Le tag url peut avoir un ou plusieurs tag fils « param ». Le tag param permet de préciser un parametre et sa valeur pour qu'il soit ajouté à l'url générée.

Le tag param possède deux attributs :

Attribut	Rôle
name	nom du paramètre
value	valeur du paramètre

Exemple :

```
<a href="<c:url url="/index.jsp"/>"/>
```

37.4. La bibliothèque XML

Cette bibliothèque permet de manipuler des données en provenance d'un document XML.

Elle propose les tags suivants répartis dans trois catégories :

Catégorie	Tag
Fondamentale	parse set out
Gestion du flux (condition et itération)	if choose forEach
Transformation XSLT	transform

Les exemples de cette section utilisent un fichier xml nommé personnes.xml dont le contenu est le suivant :

Fichier utilisé dans les exemples :

```
<personnes>
  <personne id="1">
    <nom>nom1</nom>
    <prenom>prenom1</prenom>
  </personne>
  <personne id="2">
    <nom>nom2</nom>
    <prenom>prenom2</prenom>
  </personne>
```

```
  <personne id="3">
    <nom>nom3</nom>
    <prenom>prenom3</prenom>
  </personne>
</personnes>
```

L'attribut select des tags de cette bibliothèque utilise la norme Xpath pour sa valeur. JSTL propose une extension supplémentaire à Xpath pour préciser l'objet sur lequel l'expression doit être évaluée. Il suffit de préfixer le nom de la variable par un $

Exemple : recherche de la personne dont l'id est 2 dans un objet nommé listepersonnes qui contient l'arborescence du document xml.

```
$listepersonnes/personnes/personne[@id=2]
```

L'implémentation de JSTL fournie avec le JWSDP utilise Jaxen comme moteur d'interprétation XPath. Donc pour utiliser cette bibliothèque, il faut s'assurer que les fichiers saxpath.jar et jaxen−full.jar soient présents dans le répertoire lib du répertoire WEB−INF de l'application web.

Pour utiliser cette bibliothèque, il faut la déclarer dans le fichier web.xml du répertoire WEB−INF de l'application web.

Exemple :

```
  <taglib>
    <taglib−uri>http://java.sun.com/jstl/xml</taglib−uri>
    <taglib−location>/WEB−INF/tld/x.tld</taglib−location>
  </taglib>
```

Dans chaque JSP qui utilise un ou plusieurs tags de la bibliothèque, il faut la déclarer avec une directive taglib.

Exemple :

```
<%@ taglib uri="http://java.sun.com/jstl/xml" prefix="x" %>
```

37.4.1. Le tag parse

La tag parse permet d'analyser un document et de stocker le résultat dans une variable qui pourra être exploité par la JSP ou une autre JSP selon la portée sélectionnée pour le stockage.

Attribut	Rôle
xml	contenu du document à analyser
var	nom de la variable qui va contenir l'arbre DOM générer par l'analyse
scope	portée de la variable qui va contenir l'arbre DOM
varDom	
scopeDom	
filter	
System	

Exemple : chargement et sauvegarde dans une variable de la JSP

```
<c:import url="/personnes.xml" var="personnes" />
<x:parse xml="${personnes}" var="listepersonnes"  />
```

Dans cet exemple, il suffit simplement que le fichier personnes.xml soit dans le dossier racine de l'application web.

37.4.2. Le tag set

Le tag set est équivalent au tag set de la bibliothèque core. Il permet d'évaluer l'expression Xpath fournie dans l'attribut select et de placer le résultat de cette évaluation dans une variable. L'attribut var permet de préciser la variable qui va recevoir le résultat de l'évaluation sous la forme d'un noeud de l'arbre du document XML.

Il possède plusieurs attributs :

Attribut	Rôle
select	expression XPath à évaluer
var	nom de la variable qui va stocker le résultat de l'évaluation
scope	portée de la variable qui va stocker le résultat

Exemple : obtenir les informations de la personne dont l'id est 2

```
<c:import url="/personnes.xml" var="personnes" />
<x:parse xml="${personnes}" var="listepersonnes"  />
<x:set var="unepersonne" select="$listepersonnes/personnes/personne[@id=2]" />
<h1>nom = <x:out select="$unepersonne/nom"/></h1>
```

37.4.3. Le tag out

Le tag out est équivalent au tag out de la bibliothèque core. Il est permet d'évaluer l'expression Xpath fournie dans l'attribut select et d'envoyer le résultat dans le flux de sortie. L'attribut select permet de préciser l'expression Xpath qui doit être évaluée.

Il possède plusieurs attributs :

Attribut	Rôle
select	expression XPath à évaluer
escapeXML	

Exemple : Afficher le nom de la personne dont l'id est 2

```
<c:import url="/personnes.xml" var="personnes" />
<x:parse xml="${personnes}" var="listepersonnes"  />
<x:set var="unepersonne" select="$listepersonnes/personnes/personne[@id=2]" />
<h1><x:out select="$unepersonne/nom"/></h1>
```

Pour stocker le résultat de l'évaluation d'une expression dans une variable, il faut utiliser une combinaison du tag x:out et c:set

Exemple :

```
<c:set var="personneId">
      <x:out select="$listepersonnes/personnes/personne[@id=2]" />
```

```
    </c:set>
```

37.4.4. Le tag if

Ce tag est équivalent au tag if de la bibliothèque core sauf qu'il évalue une expression XPath

Il possède plusieurs attributs :

Attribut	Rôle
select	expression XPath à évaluer sous la forme d'un booléen
var	nom de la variable qui va stocker le résultat de l'évaluation
scope	portée de la variable qui va stocker le résultat de l'évaluation

37.4.5. Le tag choose

Ce tag est équivalent au tag choose de la bibliothèque core sauf qu'il évalue des expressions XPath

37.4.6. Le tag forEach

Ce tag est équivalent au tag forEach de la bibliothèque Core. Il permet de parcourir les noeuds issus de l'évaluation d'une expression Xpath.

Il possède plusieurs attributs :

Attribut	Rôle
select	expression XPath à évaluer (obligatoire)
var	nom de la variable qui va contenir le noeud en cours de traitement

Exemple : parcours des personnes et affichage de l'id, du nom et du prénom

```
<c:import url="/personnes.xml" var="personnes" />
<x:parse xml="${personnes}" var="listepersonnes"  />
<x:forEach var="unepersonne" select="$listepersonnes/personnes/*">
    <c:set var="personneId">
      <x:out select="$unepersonne/@id"/>
    </c:set>
    <c:out value="${personneId}" /> - <x:out select="$unepersonne/nom"/>  
       <x:out select="$unepersonne/prenom"/> <br>
</x:forEach>
```

37.4.7. Le tag transform

Ce tag permet d'appliquer une transformation XSLT à un document XML. L'attribut xsl permet de préciser la feuille de style XSL. L'attribut optionnel xml permet de préciser le document xml.

Il possède plusieurs attributs :

Attribut	Rôle
xslt	feuille se style XSLT (obligatoire)
xml	nom de la variable qui contient le document XML à traiter

var	nom de la variable qui va recevoir le résultat de la transformation
scope	portée de la variable qui va recevoir le résultat de la transformation
xmlSystemId	
xsltSystemId	
result	

Exemple :

```
<x:transform xml='${docXml}' xslt='${feuilleXslt}'/>
```

Le document xml à traiter peut être fourni dans le corps du tag

Exemple :

```
<x:transform xslt='${feuilleXslt}'>
  <personnes>
    <personne id="1">
      <nom>nom1</nom>
      <prenom>prenom1</prenom>
    </personne>
    <personne id="2">
      <nom>nom2</nom>
      <prenom>prenom2</prenom>
    </personne>
    <personne id="3">
      <nom>nom3</nom>
      <prenom>prenom3</prenom>
    </personne>
  </personnes>
</x:transform>
```

Le tag transform peut avoir un ou plusieurs noeuds fils param pour fournir des paramètres à la feuille de style XSLT.

37.5. La bibliothèque I18n

Cette bibliothèque facilite l'internationnalisation d'une page JSP.

Elle propose les tags suivants répartis dans trois catégories :

Catégorie	Tag
Définition de la langue	setLocale
Formattage de messages	>bundle message setBundle
Formattage de dates et nombres	formatNumber parseNumber formatDate parseDate setTimeZone timeZone

Pour utiliser cette bibliothèque, il faut la déclarer dans le fichier web.xml du répertoire WEB−INF de l'application web.

Exemple :

```
  <taglib>
    <taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
<taglib-location>/WEB-INF/tld/fmt.tld</taglib-location>
</taglib>
```

Dans chaque JSP qui utilise un ou plusieurs tags de la bibliothèque, il faut la déclarer avec une directive taglib

Exemple :

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
```

Le plus simple pour mettre en oeuvre la localisation des messages, c'est de définir un ensemble de fichier qui sont appelé bundle en anglais.

Il faut définir un fichier pour la langue par défaut et un fichier pour chaque langue particulière. Tous ces fichiers ont un préfix commun appelé basename et doivent avoir comme extension .properties. Les fichiers pour les langues particulieres doivent le préfix commun suivit d'un underscore puis du code langue et eventuellement d'un underscore suivi du code pays. Ces fichiers doivent être inclus dans le classpath : le plus simple est de les copier dans le répertoire WEB-INF/classes de l'application web.

Exemple :

```
message.properties
messageen.properties
```

Dans chaque fichier, les clés sont identiques, seul la valeur associée à la clé change.

Exemple : le fichier message.properties pour le français (langue par défaut)

```
msg=bonjour
```

Exemple : le fichier messageen.properties pour l'anglais

```
msg=Hello
```

Pour plus d'information, voir le chapitre sur l'internationalisation.

37.5.1. Le tag bundle

Ce tag permet de préciser un bundle à utiliser dans les traitements contenus dans son corps.

Il possède plusieurs attributs :

Attribut	Rôle
baseName	nom de base de ressource à utiliser (obligatoire)
prefix	

Exemple :

```
<fmt:bundle basename="message" >
  <fmt:message key="msg"/>
</fmt:bundle>
```

37.5.2. Le tag setBundle

Ce tag permet de forcer le bundle à utiliser par défaut.

Il possède plusieurs attributs :

Attribut	Rôle
baseName	nom de base de ressource à utiliser (obligatoire)
var	nom de la variable qui va stocker le nouveau bundle
scope	portée de la variable qui va recevoir le nouveau bundle

Exemple :

```
mon message =
<fmt:setBundle basename="message" />
  <fmt:message key="msg"/>
```

37.5.3. Le tag message

Ce tag permet de localiser un message.

Il possède plusieurs attributs :

Attribut	Rôle
key	clé du message à utiliser
bundle	bundle à utiliser
var	nom de la variable qui va recevoir le résultat du formattage
scope	portée de la variable qui va recevoir le résultat du formattage

Pour fournir chaque valeur, il faut utiliser un ou plusieurs tags fils param pour fournir la valeur correspondante.

Exemple :

```
mon message =
<fmt:setBundle basename="message" />
  <fmt:message key="msg"/>
```

Résultat :

```
mon message = bonjour
```

Si aucune valeur n'est trouvée pour la clé fournie alors le tag renvoie ???XXX ??? ou XXX représente le nom de la clé.

Exemple :

```
mon message =
<fmt:setBundle basename="message" />
  <fmt:message key="test"/>
```

Résultat :

```
mon message = ???test???
```

37.5.4. Le tag setLocale

Ce tag permet de sélectionner une nouvelle Locale.

Exemple :

```
<fmt:setLocale value="en"/>
mon message =
<fmt:setBundle basename="message" />
  <fmt:message key="msg"/>
```

Résultat :

```
mon message = Hello
```

37.5.5. Le tag formatNumber

Ce tag permet de formatter des nombres selon la locale. L'attribut value permet de préciser la valeur à formatter. L'attribut type permet de préciser le type de formattage à réaliser.

Il possède plusieurs attributs :

Attribut	Rôle
value	valeur à formatter
type	CURRENCY ou NUMBER ou PERCENT
pattern	format personnalisé
currencyCode	code de la monnaie à utiliser pour le type CURRENCY
currencySymbol	symbole de la monnaire à utiliser pour le type CURRENCY
groupingUsed	booléen pour préciser si les nombres doivent être groupés
maxIntegerDigits	nombre maximun de chiffre dans la partie entière
minIntegerDigits	nombre minimum de chiffre dans la partie entière
maxFractionDigits	nombre maximun de chiffre dans la partie décimale
minFractionDigits	nombre minimum de chiffre dans la partie décimale
var	nom de la variable qui va stocker le résultat
scope	portée de la variable qui va stocker le résultat

Exemple :

```
<c:set var="montant" value="12345.67" />
montant = <fmt:formatNumber value="${montant}" type="currency"/>
```

37.5.6. Le tag parseNumber

Ce tag permet de convertir une chaîne de caractère qui contient un nombre en une variable décimale.

Il possède plusieurs attributs :

Attribut	Rôle
value	valeur à traiter
type	CURRENCY ou NUMBER ou PERCENT
parseLocale	Locale à utiliser lors du traitement
integerOnly	booléen qui indique si le résultat doit être un entier (true) ou un flottant (false)
pattern	format personnalisé
var	nom de la variable qui va stocker le résultat
scope	portée de la variable qui va stocker le résultat

Exemple : convertir en entier un identifiant passé en paramètre de la requête

```
<fmt:parseNumber value="${param.id}" var="id"/>
```

37.5.7. Le tag formatDate

Ce tag permet de formatter des dates selon la locale.

Il possède plusieurs attributs :

Attribut	Rôle
value	valeur à formatter
type	DATE ou TIME ou BOTH
dateStyle	FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
timeStyle	FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
pattern	format personnalisé
timeZone	timeZone utilisé pour le formattage
var	nom de la variable qui va stocker le résultat
scope	portée de la variable qui va stocker le résultat

L'attribut value permet de préciser la valeur à formatter. L'attribut type permet de préciser le type de formattage à réaliser. L'attribut dateStyle permet de préciser le style du formattage.

Exemple :

```
<jsp:useBean id="now" class="java.util.Date" />
Nous sommes le <fmt:formatDate value="${now}" type="date" dateStyle="full"/>.
```

37.5.8. Le tag parseDate

Ce tag permet d'analyser une chaîne de caractères contenant une date pour créer un objet de type java.util.Date.

Il possède plusieurs attributs :

Attribut	Rôle
value	valeur à traiter
type	DATE ou TIME ou BOTH
dateStyle	FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
timeStyle	FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
pattern	format personnalisé
parseLocale	Locale utilisé pour le formattage
timeZone	timeZone utilisé pour le formattage
var	nom de la variable de type java.util.date qui va stocker le résultat
scope	portée de la variable qui va stocker le résultat

37.5.9. Le tag setTimeZone

Ce tag permet de stocker un fuseau horaire dans une variable.

Il possède plusieurs attributs :

Attribut	Rôle
value	fuseau horaire à stocker (obligatoire)
var	nom de la variable de stockage
scope	portée de la variable de stockage

37.5.10. Le tag timeZone

Ce tag permet de préciser un fuseau horraire particulier à utiliser dans son corps.

Il possède plusieurs attributs :

Attribut	Rôle
value	chaine de caractère ou objet java.util.TimeZone qui précise le fuseau horraire à utiliser

37.6. La bibliothèque Database

Cette biliothèque facilite l'accès aux bases de données. Son but n'est pas de remplacer les accès réalisés grace à des beans ou des EJB mais de fournir une solution simple mais non robuste pour accéder à des bases de données. Ceci est cependant particulièrement utile pour développer des pages de tests ou des prototypes.

Elle propose les tags suivants répartis dans deux catégories :

Catégorie	Tag
Définition de la source de données	setDataSource

Execution de requete SQL	query transaction update

Pour utiliser cette bibliothèque, il faut la déclarer dans le fichier web.xml du répertoire WEB−INF de l'application web.

Exemple :

```
  <taglib>
    <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
    <taglib-location>/WEB-INF/tld/sql.tld</taglib-location>
</taglib>
```

Dans chaque JSP qui utilise un ou plusieurs tags de la bibliothèque, il faut la déclarer avec une directive taglib

Exemple :

```
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
```

37.6.1. Le tag setDataSource

Ce tag permet de créer une connection vers la base de données à partir des données fournies dans les différents attributs du tag.

Il possède plusieurs attributs :

Attribut	Rôle
driver	nom de la classe du pilote JDBC à utiliser
source	url de la base de données à utiliser
user	nom de l'utilisateur à utiliser lors de la connection
password	mot de passe de l'utilisateur à utiliser lors de la connection
var	nom de la variable qui va stocker l'objet créé lors de la connection
scope	portée de la variable qui va stocker l'objet créé
dataSource	

Exemple : accéder à une base via ODBC dont le DNS est test

```
<sql:setDataSource driver="sun.jdbc.odbc.JdbcOdbcDriver" url="jdbc:odbc:test" user="" password=""/>
```

37.6.2. Le tag query

Ce tag permet de réaliser des requêtes de sélection sur une source de données.

Il possède plusieurs attributs :

Attribut	Rôle
sql	requête SQL à executer
var	nom de la variable qui stocke les résultats de l'exécution dela requête

scope	portée de la variable qui stocke le résultat
startRow	numéro de l'occurrence de départ à traiter
maxRow	nombre maximum d'occurrence à stocker
dataSource	connection particulière à la base de données à utiliser

L'attribut sql permet de préciser la requête à executer :

Exemple :

```
<sql:query var="reqPersonnes" sql="SELECT * FROM personnes" />
```

Le résultat de l'exécution de la requête est stocké dans un objet qui implémente l'interface `javax.servlet.jsp.jstl.sql.Result dont le nom est donné via l'attribut var`

L'interface Result possède cinq getter :

Méthode	Rôle
String[] getColumnNames()	renvoie un tableau de chaines de caractères qui contient le nom des colonnes
int getRowCount()	renvoie le nombre d'enregistrements trouvé lors de l'exécution de la requête
Map[] getRows()	renvoie une collection qui associe à chaque colonne la valeur associé pour l'occurrence en cours
Object[][] getRowsByIndex()	renvoie un tableau contenant les colonnes et leur valeur
boolean isLimitedByMaxRows()	renvoie un booléen qui indique si le résultat de la requête a été limité

Exemple : connaitre le nombre d'occurrence renvoyées par la requête

```
<p>Nombre d'enregistrement trouvé : <c:out value="${reqPersonnes.rowCount}" /></p>
```

La requête SQL peut être précisée avec l'attribut sql ou dans le corps du tag

Exemple :

```
<sql:query var="reqPersonnes" >
  SELECT * FROM personnes
</sql:query>
```

Le tag forEach de la bibliothèque core est particulièrement utile pour itérer sur chaque occurrence retournée par la requête SQL.

Exemple :

```
<TABLE border="1" CELLPadding="4" cellspacing="0">
<TR>
<td>id</td>
<td>nom</td>
<td>prenom</td>
</TR>

<c:forEach var="row" items="${reqPersonnes.rows}" >
```

```
<TR>
<td><c:out value="${row.id}" /></td>
<td><c:out value="${row.nom}" /></td>
<td><c:out value="${row.prenom}" /></td>
</TR>
</c:forEach>
</TABLE>
```

Il est possible de fournir des valeurs à la requête SQL. Il faut remplacer dans la requête SQL la valeur par le caractère ?. Pour fournir, la ou les valeurs il faut utiliser un ou plusieurs tags fils param.

Le tag param possède un seul attribut :

Attribut	Rôle
value	valeur de l'occurrence correspondante dans la requête SQL

Pour les valeurs de type date, il faut utiliser le tag dateParam.

Le tag dateParam possède plusieurs attributs :

Attribut	Rôle
value	objet de type java.util.date qui contient la valeur de la date (obligatoire)
type	format de la date : TIMESTAMP ou DATE ou TIME

Exemple :

```
<c:set var="id" value="2" />

<sql:query var="reqPersonnes" >
  SELECT * FROM personnes where id = ?
            <sql:param value="${id}" />
</sql:query>
```

37.6.3. Le tag transaction

Ce tag permet d'encapsuler plusieurs requêtes SQL dans une transaction.

Il possède plusieurs attributs :

Attribut	Rôle
dataSource	connection particulière à la base de données à utiliser
isolation	READCOMMITTED ou READUNCOMMITTED ou REPEATABLEREAD ou SERIALIZABLE

37.6.4. Le tag update

Ce tag permet de réaliser une mise à jour grace à une requête SQL sur la source de données.

Il possède plusieurs attributs :

Attribut	Rôle
sql	requête SQL à exécuter

var	nom de la variable qui stocke le nombre d'occurrence impactée par l'exécution dela requête
scope	portée de la variable qui stocke le nombre d'occurrence impactée
dataSource	connection particulière à la base de données à utiliser

Exemple :

```
<c:set var="id" value="2" />
<c:set var="nouveauNom" value="nom 2 modifié" />

<sql:update var="nbRec">
UPDATE personnes
SET nom = ?
WHERE id=?
lt;sql:param value="${nouveauNom}"/>
<sql:param value="${id}"/>
</sql:update>

<p>nb enregistrement modifiés = <c:out value="${nbRec}"/></p>
```

38. Les frameworks pour les applications web

La suite de ce chapitre sera développé dans une version future de ce document

Un framework est un ensemble de composants qui structure tout ou partie d'une application.

Dès que l'on développe des applications web uniquement avec les API servlet et JSP, il apparait évident que de nombreuses parties dans des applications différentes sont communes mais doivent être réécrites ou réutilisées à chaque fois. Ceci est en grande partie liée au fait que les API servlets et JSP sont des API de bas niveau. Elles ne proposent par exemple rien pour automatiser l'extraction des données de la reqûete HTTP et mapper leur contenu dans un objet, assurer les transitions entre les pages selon les circonstances, ...

Ce chapitre contient plusieurs sections :

- Intérêt et utilité
- Struts
- Expresso
- Barracuda
- Tapestry
- Turbine
- stxx
- WebMacro
- FreeMarker
- Velocity

38.1. Intérêt et utilité

L'intérêt majeur des frameworks est de proposer une structure identique pour toutes les applications web qui l'utilisent et de fournir des mécanismes plus ou moins sophistiqués pour assurer des tâches communes à toutes les applications web. Les framework proposent un cadre de développement pour applications reposant sur un squelette d'application et un ensemble de services plus ou moins sophistiqués pour faciliter le développement des applications.

Les frameworks sont aussi souvents accompagnés d'outils et de normes à respecter lors du développement.

Leur utilisation apporte plusieurs avantages :

- augmenter la productivité une fois le framework pris en main
- assurer la réutilisation de composants fiables

- faciliter la maintenance
- ...

Le plus gros défaut des frameworks est lié à leur complexitité : il faut un certain temps d'apprentissage pour avoir un minimum de maitrise et d'efficacité dans leur utilisation.

Le choix d'un framework est très important car l'utilisation d'un autre framework impose un travail important :

- prise en main du nouveau framework
- réécriture partiel de l'application

Il existe de nombreux frameworks open source dont le plus utilisé est Struts. Les frameworks open source ont l'avantage d'être développés par un grand nombre de personnes, ce qui permet d'avoir des frameworks relativement complet, fiable et fonctionnel. En plus, comme tout projet open source, les sources sont disponibles ce qui permet éventuellement de faire des modifications pour répondre à ces propres besoins ou ajouter des fonctionnalités.

Les frameworks utilisent ou peuvent être complétés par des moteurs de templates qui facilitent la génération de page web à partir de modèles.

38.1.1. Le modèle MVC

Le modèle MVC (Model View Controler) a été initialement développé pour le langage Smalltalk dans le but de mieux structurer une application avec une interface graphique.

Ce modèle sépare en trois parties une application :

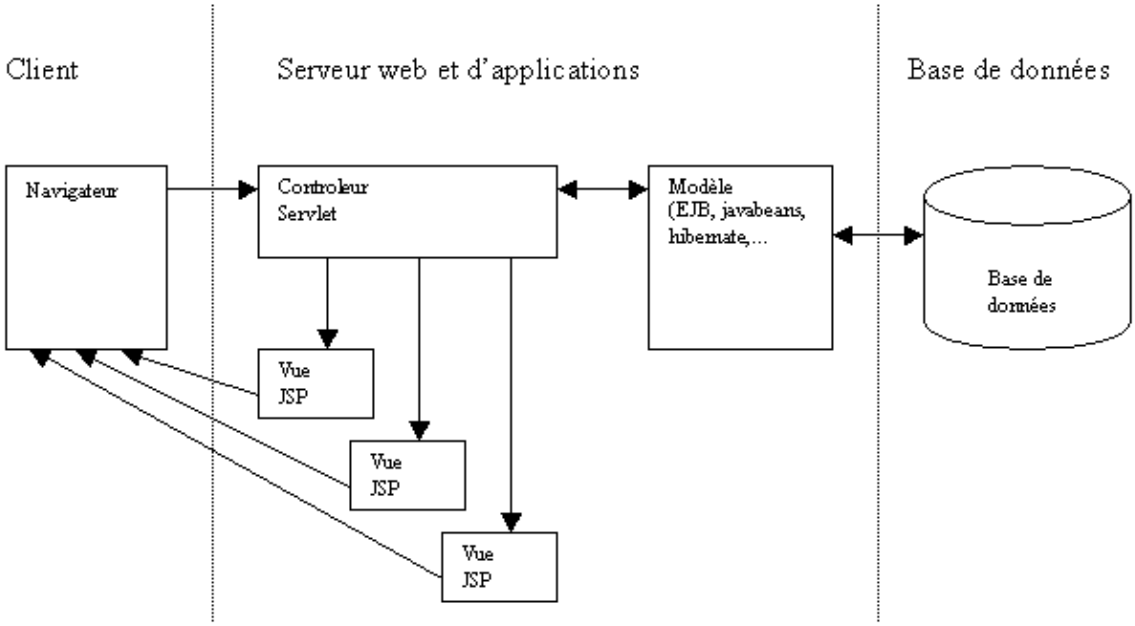
- la vue : représente l'interface homme−machine
- le modèle : représente les données de l'application
- le contrôleur : assure le lien entre les deux autres entités en contenant les règles métiers

Ce modèle peut se transposer pour les applications web : la vue est mise en oeuvre par des JSP, le contrôleur est mis en oeuvre par des servlets. Différents mécanismes peuvent être utilisés pour accéder à la base de données.

L'utilisation du modèle MVC rend un peu plus compliqué le développement de l'application qui le met en oeuvre.

38.1.2. Le modèle MVC2

Le principal défaut du modèle MVC est le nombre de servlets à développer pour une application. Pour simplifier les chose, le modèle MVC model 2 ou MVC2 propose de n'utiliser qu'une seule et unique servlet comme contrôleur. Cette servlet se charge d'assurer le worflow des traitements.



38.2. Struts

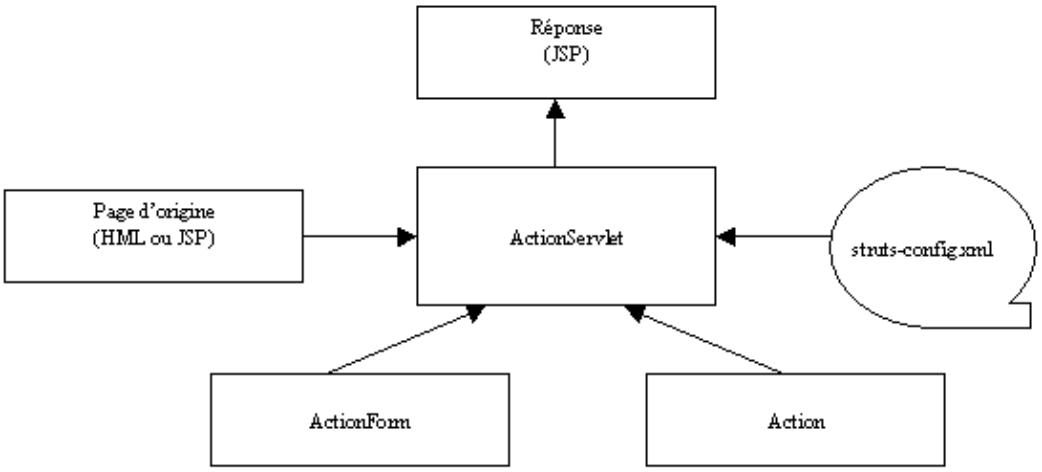
Struts est un framework pour applications web développé par le projet Jakarta de la fondation Apache. C'est la plus populaire des frameworks pour le développement d'applications web avec java .

Il a été initialement développé par Craig Mc Clanahan qui l'a donné au projet Jakarta d'Apache en mai 2000. Depuis, Struts a connu un succès grandissant auprès de la communauté du libre et des développeurs à tel point que la plupart des grands IDE propriétaires (Borland, IBM, BEA, ...) intègre une partie dédié à son utilisation.

Struts met en oeuvre le modèle MVC 2 basé sur une seule servlet et des JSP pour chaque application. L'application de ce modèle permet une séparation en trois partie distinctes de l'interface, des traitements et des données de l'application.

Strust se concentre sur la vue et le contrôleur. L'implémentation du modèle est laissée libre aux développeurs : ils ont le choix d'utiliser des java beans, un outil de mapping objet/relationnel ou des EJB.

Pour le contrôleur, Struts propose une unique servlet par application qui lit la configuration de l'application dans un fichier au format XML. Cette servlet reçoit toutes les requêtes de l'utilisateur concernant l'application. En fonction du paramètrage, il instancie un objet de type Action qui contient les traitements et renvoie une valeur particulière à la servlet. Ceci lui permet de déterminer la JSP qui affichera le résultat à l'utilisateur.



Pour la vue, Struts utilise des JSP avec un ensemble de trois bibliothèques de tags personnalisés pour faciliter leur développement.

Struts propose aussi plusieurs services techniques : pool de connexion aux sources de données, internationalisation, ...

La dernière version ainsi que toutes les informations utiles peuvent être obtenues sur le site http://jakarta.apache.org/struts/

Il existe deux versions de Struts : 1.0 et 1.1.

38.2.1. Installation et mise en oeuvre

Il faut télécharger la dernière version de Struts sur le site du projet Jakarta.

Il suffit de dezipper le fichier dans un répertoire quelconque.

Pour pouvoir utiliser Struts dans une application web, il faut copier les fichiers struts.jar et commons−*.jar dans le répertoire WEB−INF/lib de l'application.

Comme Struts met en oeuvre le modèle MVC, il est possible de développer séparement les différents composants de l'application.

38.2.2. Le développement des vues

Les vues représentent l'interface avec entre l'application et l'utilisateur. Avec le framework Struts, les vues d'une application web sont des JSP. Pour faciliter leur développement, Struts propose un ensemble de trois bibliothèques de tags personnalisés possèdant chacun un thème particulier :

- HTML
- Bean
- Logic

38.2.3. Les objets de type ActionForm

Un objet de type ActionForm est un objet qui permet à Struts de mapper automatiquement les données saisies dans une JSP avec les attributs correspondant dans l'objet.

Pour automatiser cette tache, Struts utilise l'introspection pour rechercher un accesseur correspondant au nom du paramètre contenant la donnée.

Pour chaque page contenant des données à utiliser, il faut définir un objet qui hérite de la classe org.apache.struts.action.ActionForm. Par convention, le nom de cette classe est la nom de la page suivi de "Form".

Pour chaque donnée, il faut définir un attribut privé qui contiendra la valeur, un getter et un setter public en respectant les normes de développement des java beans.

La méthode reset() doit être redéfinie pour initialiser chaque attribut.

Il faut compiler cette classe est la placer dans le répertoire WEB−INF/classes suivi de l'arborescence correspondant au package de la classe.

Il faut aussi déclarer pour chaque ActionForm, un tag form−bean dans le fichier struts−config.xml. Ce tag possède plusieurs attributs :

Attribut	Rôle
name	le nom sous lequel Struts va connaître l'objet
type	le type complètement qualifié de la classe de type ActionForm

38.2.4. Le développement de la partie contrôleur

Basée sur le modèle MVC 2, la partie contrôleur de Struts est implémentée en utilisant une seule et unique servlet par application. Cette servlet doit hériter de la classe org.apache.struts.action.ActionServlet.

Cette servlet possède des traitements génériques qui utilisent les informations contenues dans le fichier struts−config.xml et dans des objets du type org.apache.struts.action.Action

La servlet reçoit les requêtes http émises par le client et en fonction de celles ci, elle appelle un objet du type Action qui lui est associé dans le fichier struts−config.xml.

Un objet de type Action contient une partie spécifique de la logique métier de l'application. Cet objet doit étendre la classe org.apache.struts.action.Action.

La méthode la plus importante de cette classe est la méthode execute(). C'est elle qui doit contenir les traitements qui seront exécutés.

38.2.4.1. Le fichier struts−config.xml

38.3. Expresso

Expresso est un framework open source développé par JCorporate qui est basé sur Struts, depuis sa version 4.

Une des particularités de ce framework est de proposer un ensemble de fonctionnalités très intéressantes :

- outils de mapping objet−relationnel pour la persistence des données
- gestion d'un pool de connexions aux bases de données
- workflow
- identification et authentification pour la sécurité
- ...

La version 5.0 de ce framework est disponible depuis octobre 2002.

La version 5.5, publié en mai 2004 repose sur Struts 1.1.

38.4. Barracuda

Le site officiel de Barracuda est à l'url : http://barracudamvc.org/Barracuda/index.html

38.5. Tapestry

Tapestry est un framewor open−source développé par le projet Jakarta de la fondation Apache.

Le site officiel de Tapestry est à l'url : http://jakarta.apache.org/tapestry/

38.6. Turbine

Le site officiel de Turbine est à l'url : http://jakarta.apache.org/turbine/

38.7. stxx

38.8. WebMacro

Webmacro est un moteur de template open source.

Le site officiel de Webmacro est à l'url : http://www.webmacro.org/

38.9. FreeMarker

FreeMarker est un moteur de template open source développé en Java.

Le site officiel de FreeMarker est à l'url : http://freemarker.sourceforge.net/

38.10. Velocity

http://jakarta.apache.org/velocity/

39. Java Server Faces

39.1. Présentation

Les technologies permettant de développer des applications web avec Java ne cessent d'évoluer :

1. Servlets
2. JSP
3. MVC Model 1 : servlets + JSP
4. MVC Model 2 : un seule servlet + JSP
5. Java Server Faces

Java Server Faces (JSF) est une technologie dont le but est de proposer un framework qui facilite et standardise le développement d'applications web avec Java. Son développement a tenu compte des différentes expériences acquises lors de l'utilisation des technologies standards pour le développement d'applications web (servlet, JSP, JSTL) et de différents frameworks (Struts, ...).

Le grand intérêt de JSF est de proposer un framework qui puisse être mis en oeuvre par des outils pour permettre un développement de type RAD pour les applications web et ainsi faciliter le développement des applications de ce type. Ce type de développement était déjà courant pour des applications standalone ou client/serveur lourd avec des outils tel que Delphi de Borland, Visual Basic de Microsoft ou Swing avec Java.

Ce concept n'est pourtant pas nouveau dans les applications web puisqu'il est déjà mis en oeuvre par WebObject d'Apple et plus récemment par ASP.Net de Microsoft mais sa mise en oeuvre à grande échelle fût relativement tardive. L'adoption du RAD pour le développement web trouve notamment sa justification dans le coût élevé de développement de l'IHM à la « main » et souvent par copier/coller d'un mixe de plusieurs technologies (HTML, Javascript, ...), rendant fastidieu et peu fiable le développement de ces applications.

Plusieurs outils commerciaux intègrent déjà l'utilisation de JSF notamment Studio Creator de Sun, WSAD d'IBM, JBuilder de Borland, JDeveloper d'Oracle, ...

Même si JSF peut être utilisé par codage à la main, l'utilisation d'un outil est fortement recommandée pour pouvoir mettre en oeuvre rapidement toute la puissance de JSF.

Ainsi de par sa complexité et sa puissance, JSF s'adapte parfaitement au développement d'applications web complexes en facilitant leur écriture.

Les pages officielles de cette technologie sont à l'url : http://java.sun.com/j2ee/javaserverfaces/

La version 1.0 de Java Server Faces, développée sous la JSR−127 , a été validée en mars 2004.

JSF est une technologie utilisée côté serveur dont le but est de faciliter le développement de l'interface utilisateur en séparant clairement la partie « interface » de la partie « métier » d'autant que la partie interface n'est souvent pas la plus compliquée mais la plus fastidieuse à réaliser.

Cette séparation avait déjà été initiée avec la technologie JSP et particulièrement les bibliothèques de tags personnalisés. Mais JSF va encore plus loin en reposant sur le modèle MVC et en proposant de mettre en oeuvre :

- l'assemblage de composants serveur qui génèrent le code de leur rendu avec la possibilité d'associer certains

composants à une source de données encapsulée dans un bean
- l'utilisation d'un modèle de développement standardisé reposant sur l'utilisation d'événements et de listener
- la conversion et la validation des données avant leur utilisation dans les traitements
- la gestion de l'état des composants de l'interface graphique
- la possibilité d'étendre les différents modèles et de créer ces propres composants
- la configuration de la navigation entre les pages
- le support de l'internationalisation
- le support pour l'utilisation par des outils graphiques du framework afin de faciliter sa mise en oeuvre

JSF se compose :

- d'une spécification qui définit le mode de fonctionnement du framework et une API : l'ensemble des classes de l'API est contenu dans les packages javax.faces.
- d'une implémentation de référence
- de bibliothèques de tags personnalisés fournies par l'implémentation pour utiliser les composants dans les JSP, gérer les événements, valider les données saisies, ...

Le rendu des composants ne se limite pas à une seule technologie même si l'implémentation de référence ne propose qu'un rendu des composants en HTML.

Le traitement d'une requête traitée par une application utilisant JSF utilise un cycle de vie particulier constitué de plusieurs étapes :

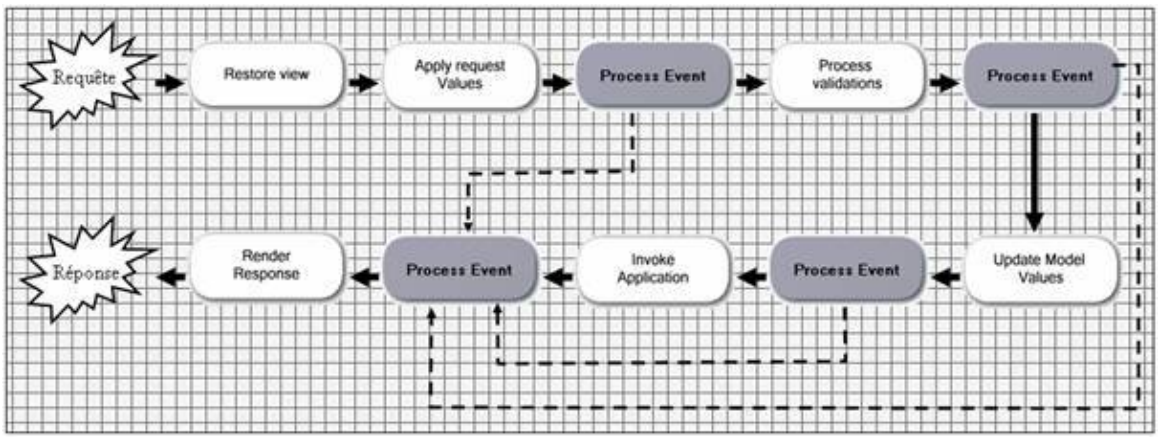
- Création de l'arbre de composants
- Extraction des données des différents composants de la page
- Conversion et validation des données
- Extraction des données validées et mise à jour du modèle de données (javabean)
- Traitements des événements liés à la page
- Génération du rendu de la réponse

Ces différentes étapes sont transparentes lors d'une utilisation standard de JSF.

39.2. Le cycle de vie d'une requête

JSF utilise la notion de vue (view) qui est composée d'une arborescence ordonnée de composants inclus dans la page.

Les requêtes sont prises en charge et gérées par le contrôleur d'une application JSF (en général une servlet). Celle ci va assurer la mise en oeuvre d'un cycle de vie des traitements permettant de traiter la requête en vue d'envoyer une réponse au client.



JSF propose pour chaque page un cycle de vie pour traiter la requête HTTP et générer la réponse. Ce cycle de vie est composé de plusieurs étapes :

1. Restore view ou Reconstruct Component Tree : cette première phase permet au serveur de recréer l'arborescence des composants qui composent la page. Cette arborescence est stockée dans un objet de type FacesContext et

sera utilisée tout au long du traitement de la requête.
2. Apply Request Value : dans cette étape, les valeurs des données sont extraites de la requête HTTP pour chaque composant et sont stockées dans leur composant respectif dans le FaceContext. Durant cette phase des opérations de convertions sont réalisées pour permettre de transformer les valeurs stockées sous forme de chaîne de caractères dans la requête http en un type utilisé pour le stockage des données.
3. Perform validations : une fois les données extraites et converties, il est possible de procéder à leur validation en appliquant les validators enregistrés auprès de chaque composant. Les éventuelles erreurs de convertions sont stockées dans le FaceContext. Dans ce cas, l'étape suivante est directement « Render Response » pour permettre de réafficher la page avec les valeurs saisies et afficher les erreurs
4. Synchronize Model ou update model values : cette étape permet de stocker dans les composants du FaceContext leur valeur locale validée respective. Les éventuelles erreurs de convertions sont stockées dans le FaceContext. Dans ce cas, l'étape suivante est directement « Render Response » pour permettre de réafficher la page avec les valeurs saisies et afficher les erreurs
5. Invoke Application Logic : dans cette étape, le ou les événements émis dans la page sont traités. Cette phase doit permettre de déterminer quelle sera la page résultat qui sera renvoyée dans la réponse en utilisant les règles de navigation définie dans l'application. L'arborescence des composants de cette page est créée.
6. Render Response : cette étape se charge de créer le rendue de la page de la réponse.

39.3. Les implémentations

Java Server Faces est une spécification : il est donc nécessaire d'obtenir une implémentation de la part d'un tiers.

Plusieurs implémentations commerciales ou libres sont disponibles, notamment l'implémentation de référence de Sun et MyFaces qui est devenu un projet du groupe Apache.

39.3.1. L'implémentation de référence

Comme pour toute JSR validée, Sun propose une implémentation de référence des spécifications de la JSR , qui soit la plus complète possible.

Plusieurs versions de l'implémentation de référence de Sun sont proposées :

Version	Date de diffusion
1.0	Mars 2004
1.1	Mai 2004
1.1_01	Septembre 2004

La solution la plus simple pour utiliser l'implémentation de référence est d'installer le JWSDK 1.3 qui est fourni en standard avec l'implémentation de référence de JSF. La version de JSF fournie avec le JWSDK 1.3 est la 1.0.

Pour utiliser la version 1.1, il faut supprimer le répertoire jsf dans le répertoire d'installation de JWSDK, télécharger l'implémentation de référence, décompresser son contenu dans le répertoire d'installation de JWSDK et renommer le répertoire jsf−1_1_01 en jsf.

Il est aussi possible de télécharger l'implémentation de référence sur le site de Sun et de l'installer « manuellement » dans un conteneur web tel que Tomcat. Cette procédure sera détaillée dans une des sections suivantes.

Pour cela, il faut télécharger le fichier jsf−1_1_01.zip et le décompresser dans un répertoire du système. L'archive contient les bibliothèques de l'implémentation, la documentation des API et des exemples.

Les exemples de ce chapitre vont utiliser cette version 1.1 de l'implémentation de référence des JSF.

39.3.2. MyFaces

MyFaces est une implémentation libre des Java Server Faces qui est devenu un projet du groupe Apache.

Il propose en plus plusieurs composants spécifiques en plus de ceux imposés par les spécifications JSF.

Le site de MyFaces est à l'url : http://myfaces.apache.org/

Il faut télécharger le fichier et le décompresser dans un répertoire du système. Il suffit alors de copier le fichier myfaces−examples.war dans le répertoire webapps de Tomcat. Relancez Tomcat et saisie l'url http://localhost:8080/myfaces−examples

Pour utiliser MyFaces dans ses propres applications, il faut réaliser plusieurs opérations.

Il faut copier les fichiers *.jar du répertoire lib de MyFaces et myfaces−jsf−api.jar dans le répertoire WEB−INF/lib de la webapp.

Dans chaque page qui va utiliser les composants de MyFaces, il faut déclarer la bibliothèque de tags dédiés.

Exemple :
`<%@ taglib uri="http://myfaces.sourceforge.net/tld/myfaces_ext_0_9.tld" prefix="x"%>`

39.4. Configuration d'une application

Les applications utilisant JSF sont des applications web qui doivent respecter les spécifications de J2EE.

En tant que telle, elles doivent avoir la structure définie par J2EE pour toutes les applications web :

/
/WEB−INF
/WEB−INF/web.xml

/WEB−INF/lib
/WEB−INF/classes

Le fichier web.xml doit contenir au minimum certaines informations notamment, la servlet faisant office de contrôleur, le mapping des url pour cette servlet et des paramètres.

Exemple :

```
<!DOCTYPE web−app PUBLIC
"−//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web−app_2_3.dtd">
<web−app>
  <display−name>Test JSF</display−name>
  <description>Application de tests avec JSF</description>
  <context−param>
    <param−name>javax.faces.STATE_SAVING_METHOD</param−name>
    <param−value>client</param−value>
  </context−param>
  <!−− Faces Servlet −−>
  <servlet>
    <servlet−name>Faces Servlet</servlet−name>
    <servlet−class>javax.faces.webapp.FacesServlet</servlet−class>
    <load−on−startup> 1 </load−on−startup>
  </servlet>
  <!−− Faces Servlet Mapping −−>
  <servlet−mapping>
    <servlet−name>Faces Servlet</servlet−name>
    <url−pattern>*.jsf</url−pattern>
  </servlet−mapping>
</web−app>
```

Chaque implémentation nécessite un certain nombre de bibliothèques tiers pour leur bon fonctionnement.

Par exemple, pour l'implémentation de référence, les bibliothèques suivantes sont nécessaires :

jsf−api.jar
jsf−ri.jar
jstl.jar
standard.jar
common−beanutils.jar
commons−digester.jar
commons−collections.jar
commons−logging.jar

Remarque : avec l'implémentation de référence, il n'y a aucun fichier .tld à copier car ils sont intégrés dans le fichier jsf−impl.jar.

Les fichiers nécessaires dépendent de l'implémentation utilisée.

Ces bibliothèques peuvent être mises à disposition de l'application selon plusieurs modes :

- incorporées dans le package de l'application dans le répertoire /WEB−INF/lib
- incluses dans le répertoire des bibliothèques partagées par les applications web des conteneurs web s'ils proposent une telle fonctionnalité. Par exemple avec Tomcat, il est possible de copier ces bibliothèques dans le répertoire shared/lib.

L'avantage de la première solution est de faciliter la portabilité de l'application sur différents conteneur web mais elle duplique ces fichiers si plusieurs applications utilisent JSF.

Les avantages et inconvénients de la première solution sont exactement l'opposé de la seconde solution. Le choix de l'une ou l'autre est donc à faire en fonction du contexte de déploiement.

39.5. La configuration de l'application

Toute application utilisant JSF doit posséder au moins deux fichiers de configuration qui vont contenir les informations nécessaires à la bonne configuration et exécution de l'application.

Le premier fichier est le descripteur de toute application web J2EE : le fichier web.xml contenu dans le répertoire WEB−INF.

Le second fichier est un fichier de configuration particulier au paramétrage de JSF au format XML nommé faces−config.xml.

39.5.1. Le fichier web.xml

Le fichier web.xml doit contenir au minimum certaines informations notamment, la servlet faisant office de contrôleur, le mapping des urls pour cette servlet et des paramètres pour configurer JSF.

Exemple :

```
<!DOCTYPE web−app PUBLIC
"−//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web−app_2_3.dtd">
<web−app>
  <display−name>Test JSF</display−name>
  <description>Application de tests avec JSF</description>
  <context−param>
    <param−name>javax.faces.STATE_SAVING_METHOD</param−name>
    <param−value>client</param−value>
  </context−param>

  <!−− Servlet faisant office de controleur−−>
  <servlet>
    <servlet−name>Faces Servlet</servlet−name>
    <servlet−class>javax.faces.webapp.FacesServlet</servlet−class>
    <load−on−startup> 1 </load−on−startup>
  </servlet>

  <!−Le mapping de la servlet −−>
  <servlet−mapping>
    <servlet−name>Faces Servlet</servlet−name>
    <url−pattern>*.faces</url−pattern>
  </servlet−mapping>
</web−app>
```

Le tag <servlet> permet de définir une servlet et plus particulièrement dans ce cas de préciser la servlet qui sera utilisée comme contrôleur dans l'application. Le plus simple est d'utiliser la servlet fournie avec l'implémentation de référence javax.faces.webapp.FacesServlet. Le tag <load−on−startup> avec comme valeur 1 permet de demander le chargement de cette servlet au lancement de l'application.

Le tag <servlet−mapping> permet de préciser le mapping des urls qui seront traitées par la servlet. Ce mapping peut prendre deux formes :

- mapping par rapport à une extension : exemple <url−pattern>*.faces</url−pattern>.
- mapping par rapport à un préfixe : exemple <url−pattern>/faces/*</url−pattern>.

Les URL utilisées pour des pages mettant en oeuvre JSF doivent obligatoirement passer par cette servlet. Ces urls peuvent être de deux formes selon le mapping défini.

Exemple :

- http://localhost:8080/nom_webapp/index.faces
- http://localhost:8080/nom_webapp/faces/index.jsp

Dans les deux cas, c'est la servlet utilisée comme contrôleur qui va déterminer le nom de la page JSP à utiliser.

Le paramètre de contexte javax.faces.STATE_SAVING_METHOD permet de préciser le mode d'échange de l'état de l'arbre des composants de la page. Deux valeurs sont possibles :

- client :
- server :

Il est possible d'utiliser l'extension .jsf pour les fichiers JSP utilisant JSF à condition de correctement configurer le fichier web.xml dans ce sens. Pour cela deux choses sont à faire :

- il faut demander le mapping des url terminant par .jsf par la servlet

  <servlet−mapping>
  <servlet−name>jsp</servlet−name>
  <url−pattern>*.jsf</url−pattern>
  </servlet−mapping>
- il faut préciser à la servlet le suffix par défaut à utiliser

  <context−param>
  <param−name>javax.faces.DEFAULT_SUFFIX</param−name>
  <param−value>.jsf</param−value>
  </context−param>

Le démarrage d'une application directement avec une page par défaut utilisant JSF ne fonctionne pas correctement. Il est préférable d'utiliser une page HTML qui va effectuer une redirection vers la page d'accueil de l'application

Exemple :

```
<html>
  <head>
    <meta http−equiv="Refresh" content= "0; URL=index.faces"/>
    <title>Demarrage de l'application</title>
  </head>
  <body>
    <p>D&eacute;marrage de l'application ...</p>
  </body>
</html>
```

Il suffit alors de préciser dans le fichier web.xml que cette page est la page par défaut de l'application.

Exemple :

```
...
<welcome−file−list>
  <welcome−file>index.htm</welcome−file>
</welcome−file−list>
...
```

39.5.2. Le fichier faces−config.xml

Le plus simple est de placer ce fichier dans le répertoire WEB−INF de l'application Web.

Il est aussi possible de préciser son emplacement dans un paramètre de contexte nommé javax.faces.application.CONFIG_FILES dans le fichier web.xml. Il est possible par ce biais de découper le fichier de configuration en plusieurs morceaux. Ceci est particulièrement intéressant pour de grosses applications car un seul fichier de configuration peut dans ce cas devenir très gros. Il suffit de préciser chacun des fichiers séparés par une virgule dans le tag <param−value>.

Exemple :

```
...
<context-param>
  <param-name>javax.faces.application.CONFIG_FILES</param-name>
  <param-value>
/WEB-INF/ma-faces-config.xml, /WEB-INF/navigation-faces.xml, /WEB-INF/beans-faces.xml
  </param-value>
</context-param>
...
```

Ce fichier au format XML permet de définir et de fournir des valeurs d'initialisation pour des ressources nécessaires à l'application utilisant JSF.

Ce fichier doit impérativement respecter la DTD proposée par les spécifications de JSF :

http://java.sun.com/dtd/web-facesconfig_1_0.dtd

Le tag racine du document XML est le tag <face-config>. Ce tag peut avoir plusieurs tags fils :

Tag	Rôle
application	permet de préciser ou de remplacer des éléments de l'application
factory	permet de remplacer des fabriques par des fabriques personnalisées de certaines ressources (FacesContextFactory, LifeCycleFactory, RenderKitFactory, ...)
component	définit un composant graphique personnalisé
converter	définit un convertisseur pour encoder/décoder les valeurs des composants graphiques (conversion de String en Object et vice et versa)
managed-bean	définit un objet utilisé par un composant qui est automatiquement créé, initialisé et stocké dans une portée précisée
navigation-rule	définit les règles qui permettent de déterminer l'enchaînement des traitements de l'application
referenced-bean	
render-kit	définit un kit pour le rendu des composants graphiques
lifecycle	
validator	définit un validateur personnalisé de données saisies dans un composant graphique

Ces tags fils peuvent être utilisé 0 ou plusieurs fois dans le tag <face-config>.

Le tag <application> permet de préciser des informations sur les entités utilisées par l'internationalisation et/ou de remplacer des éléments de l'application.

Les éléments à remplacer peuvent être : ActionListener, NavigationHandler, ViewHandler, PropertyResolver, VariableResolver. Ceci n'est utile que si la version fournie dans l'implémentation ne correspond pas aux besoins et doit être personnalisée par l'écriture d'une classe dédiée.

Le tag fils <message-bundle> permet de préciser le nom de base des fichiers de ressources utiles à l'internationnalisation.

Le tag <locale-config> permet de préciser quelles sont les locales qui sont supportées par l'application. Il faut utiliser autant de tag fils <supported-locale> que de locales supportées. Le tag fil <default-locale> permet de préciser la locale par défaut.

Exemple :

```
...
<application>
  <message-bundle>com.moi.test.jsf.monapp.bundles.Messages</message-bundle>
```

```
  <locale-config>
    <default-locale>fr</default-locale>
    <supported-locale>en</supported-locale>
  </locale-config>
</application>
...
```

39.6. Les beans

Les beans sont largement utilisées dans une application utilisant JSF notamment pour permettre l'échange de données entre les différentes entités et le traitement des événements.

Les beans sont des classes qui respectent une spécification particulière notamment la présence :

- de getters et de setters qui respectent une convention de nommage particulière pour les attributs
- un constructeur par défaut sans arguments

39.6.1. Les beans managés (managed bean)

Les beans managés sont des javabeans dont le cycle de vie va être géré par le framework JSF en fonction des besoins et du paramétrage fourni dans le fichier de configuration.

Dans le fichier de configuration, chacun de ces beans doit être déclaré avec un tag <managed-bean>. Ce tag possède trois tags fils obligatoires :

- <managed-bean-name> : le nom attribué au bean (celui qui sera utilisé lors de son utilisation)
- <managed-bean-class> : le type pleinenement qualifié de la classe du bean
- <managed-bean-scope> : précise la portée dans laquelle le bean sera stockée et donc utilisable

La portée peut prendre les valeurs suivantes :

- request : cette portée est limitée entre l'emission de la requête et l'envoi de la réponse. Les données stockées dans cette portée sont utilisables lors d'un transfert vers une autre page (forward). Elles sont perdues lors d'une redirection (redirect).
- session : cette portée permet l'échange de données entre plusieurs échanges avec un même client
- application : cette portée permet l'accès à des données pour toutes les pages d'une même application quelque soit l'utilisateur

Exemple :

```
...
<managed-bean>
  <managed-bean-name>login</managed-bean-name>
  <managed-bean-class>com.jmd.test.jsf.LoginBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
...
```

Il est possible de fournir des valeurs par défaut aux propriétés en utilisant le tag <managed-property>. Ce tag possède deux tags fils :

- <property-name> : nom de la propriété du bean
- <value> : valeur à associer à la propriété

Exemple :

```
...
<managed-bean>
  <managed-bean-name>login</managed-bean-name>
  <managed-bean-class>com.jmd.test.jsf.LoginBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>nom</property-name>
    <value>test</value>
  </managed-property>
</managed-bean>
...
```

Lorsque que le bean sera instancié, JSF appellera automatiquement les setters des propriétés identifiées dans des tags <managed−property> avec les valeurs fournies dans leur tag <value> respectif.

Pour initialiser la propriété à null, il faut utiliser le tag <null−value>

Exemple :

```
...
<managed-property>
  <property-name>nom</property-name>
  <null-value>
</managed-property>
...
```

Ces informations seront utilisées par JSF pour automatiser la création ou la récupération d'un bean lorsque celui ci sera utilisé dans l'application.

Le grand intérêt de ce mécanisme est de ne pas avoir à se soucier de l'instanciation du bean ou de sa recherche dans la portée puisque c'est le framework qui va s'en occuper de façon transparente.

39.6.2. Les expressions de liaison de données d'un bean

Il est toujours nécessaire dans la partie présentation d'obtenir la valeur d'une donnée d'un bean pour par exemple l'afficher.

JSF propose une syntaxe basée sur des expressions qui facilite l'utilisation des valeurs d'un bean. Ces expressions doivent être délimitées par #{ et }.

Basiquement une expression est composée du nom du bean suivi du nom de la propriété désirée séparés par un point.

Exemple :

```
<h:inputText value="#{login.nom}"/>
```

Cet exemple affecte la valeur de l'attribut nom du bean login au composant de type saisie de texte. Dans ce cas précis, c'est aussi cet attribut de ce bean qui recevra la valeur saisie lorsque la page sera envoyée au serveur.

En fonction du contexte le résultat de l'évaluation peut conduire à l'utilisation du getter (par exemple pour afficher la valeur) ou du setter (pour affecter la valeur après un envoi de la page). C'est JSF qui le détermine en fonction du contexte.

La notation par point peut être remplacée par l'utilisation de crochets. Dans ce cas, le nom de la propriété doit être mis entre simples ou doubles quotes dans les crochets.

Exemple :

```
login.nom
login["nom"]
login['nom']
```

Ces trois expressions sont rigoureusement identiques. Cette syntaxe peut être plus pratique lors de la manipulation de collections mais elle est obligatoire lorsque la propriété contient un point.

Exemple :

```
msg["login.titre"]
```

L'utilisation des quotes simples ou doubles est équivalente car il faut les imbriquer par exemple lors de leur utilisation comme valeur de l'attribut d'un composant.

Exemple :

```
<h:inputText value='#{login["nom"]}'/>
<h:inputText value="#{login['nom']}"/>
```

Attention, la syntaxe utilisée par JSF est proche mais différente de celle proposée par JSTL : JSF utilise le délimiteur #{ ... } et JSTL utilise le délimiteur ${ ... } .

JSF définit un ensemble de variables pré−définies, utilisables dans les expressions de liaison de données :

Variables	Rôle
header	une collection de type Map encapsulant les éléments définis dans les paramètres de l'en−tête de la requête http (seule la première valeur est renvoyée)
header−value	une collection de type Map encapsulant les éléments définis dans les paramètres de l'en−tête de la requête http (toutes les valeurs sont renvoyées sous la forme d'un tableau)
param	une collection de type Map encapsulant les éléments définis dans les paramètres de la requête http (seule la première valeur est renvoyée)
param−values	une collection de type Map encapsulant les éléments définis dans les paramètres de la requête http (toutes les valeurs sont renvoyées sous la forme d'un tableau)
cookies	une collection de type Map encapsulant les éléments définis dans les cookies
initParam	une collection de type Map encapsulant les éléments définis dans les paramètres d'initialisation de l'application
requestScope	une collection de type Map encapsulant les éléments définis dans la portée request
sessionScope	une collection de type Map encapsulant les éléments définis dans la portée session
applicationScope	une collection de type Map encapsulant les éléments définis dans la portée application
facesContext	une instance de la classe FacesContext
View	une instance de la classe UIViewRoot qui encapsule la vue

Lorsque qu'une variable est utilisée dans une expression, JSF recherche dans la liste des variables pré−définies, puis recherche une instance dans la portée request, puis dans la portée session et enfin dans la portée application. Si aucune instance n'est trouvée, alors JSF crée une nouvelle instance en tenant compte des informations du fichier de configuration. Cette instanciation est réalisée par un objet de type VariableResolver de l'application.

La syntaxe des expressions possède aussi quelques opérateurs :

Opérateurs	Rôle	Exemple

+ − * / % div mod	opérateurs arithmétiques	
< <= > >= == != lt le gt ge eq ne	opérateurs de comparaisons	
&& || ! and or not	opérateurs logiques	<h:inputText rendered="#{!monBean.affichable}" />
Empty	opérateur vide : un objet null, une chaîne vide, un tableau ou une collection sans élement,	
? :	opérateur ternaire de test	

Il est possible de concaténer le résultat de l'évaluation de plusieurs expressions simplement en les plaçant les uns à la suite des autres.

Exemple :

```
<h:outputText value="#{messages.salutation}, #{utilisateur.nom}!"/>
```

Il est parfois nécessaire d'évaluer une expression dans le code des objets métiers pour obtenir sa valeur. Comme tous les composants sont stockés dans le FaceContext, il est possible d'accéder à cet objet pour obtenir les informations désirées. Il est d'abord nécessaire d'obtenir l'instance courante de l'objet FaceContext en utilisant la méthode statique getCurrentInstance().

Exemple :

```
FacesContext context = FacesContext.getCurrentInstance();
ValueBinding binding = context.getApplication().createValueBinding("#{login.nom}");
String nom = (String) binding.getValue(context);
```

39.6.3. Backing bean

Les beans de type backing bean sont spécialement utilisés avec JSF pour encapsuler tout ou partie des composants qui composent une page et ainsi faciliter leur accès notamment lors des traitements.

Ces beans sont particulièrement utiles durant des traitements réalisés lors de validations ou de traitements d'événements car ils permettent un accès aux composants dont ils possèdent une référence.

Exemple :

```
package com.jmd.test.jsf;

import javax.faces.component.UIInput;

public class LoginBean {

  private UIInput composantNom;
  private String nom;
  private String mdp;

  public UIInput getComposantNom() {
    return composantNom;
  }

  public void setComposantNom(UIInput input) {
    composantNom = input;
  }
```

```
  public String getNom() {
    return nom;
  }
...
}
```

Dans la vue, il est nécessaire de lier un composant avec son attribut correspondant dans le backing bean. L'attribut binding d'un composant permet de réaliser cette liaison.

Exemple :

```
<h:inputText value="#{login.nom}" binding="#{login.composantNom}" />
```

39.7. Les composants pour les interfaces graphiques

JSF propose un ensemble de composants serveurs pour faciliter le développement d'interfaces graphiques utilisateur.

Pour les composants, JSF propose :

- un ensemble de classes qui gèrent le comportement et l'état d'un composant
- un modèle pour assurer le rendu du composant pour un type d'application (par exemple HTML)
- un modèle de gestion des événements émis par le composant reposant sur le modèle des listeners
- la possibilité d'associer à un composant un composant de conversion de données ou de validation des données

Tous ces composants héritent de la classe abstraite UIComponentBase.

JSF propose 12 composants de base :

UICommand	Composant qui permet de réaliser une action qui lève un événement
UIForm	Composant qui regroupe d'autres composants dont l'état sera renvoyé au serveur lors de l'envoi au serveur
UIGraphic	Composant qui représente une image
UIInput	Composant qui permet de saisir des données
UIOutput	Composant qui permet d'afficher des données
UIPanel	Composant qui regroupe d'autre composant à afficher sous la forme d'un tableau
UIParameter	
UISelectItem	Composant qui représente un élément sélectionné parmi un ensemble d'éléments
UISelectItems	Composant qui représente un ensemble d'éléments
UISelectBoolean	Composant qui permet de sélectionner parmis deux états
UISelectMany	Composant qui permet de sélectionner plusieurs éléments parmi un ensemble
UISelectOne	Composant qui permet de sélectionner un seul élément parmi un ensemble

Ces classes sont des javabeans qui définissent les fonctionnalités de base des composants permettant la saisie et la sélection de données.

Chacun de ces composants possède un type, un identifiant, une ou plusieurs valeurs locales et des attributs. Ils sont extensibles et il est même possible de créer ces propres composants.

Le comportement de ces composants repose sur le traitement d'évenements respectement le modèle de gestion des événements de JSF.

Ces classes ne sont pas utilisées directement : elles sont utilisées par la bibliothèque de tags personnalisés qui se charge de les instancier et de leur associer le modèle de rendu adéquat.

Ces classes ne prennent pas en charge le rendu du composant. Par exemple, un objet de type UICommand peut être rendu en HTML sous la forme d'un lien hypertexte ou d'un bouton de formulaire.

39.7.1. Le modèle de rendu des composants

Pour chaque composant, il est possible de définir un ou plusieurs modèles qui se chargent du rendu d'un composant dans un contexte client particulier (par exemple HTML).

L'association entre un composant et son modèle de rendu est réalisée dans un RenderKit : il précise pour chaque composant quel est le ou les modèles de rendu à utiliser. Par exemple, un objet de type UISelectOne peut être rendu sous la forme d'un ensemble de bouton radio, d'une liste ou d'une liste déroulante. Chacun de ces rendus est définis par un objet de type Renderer.

L'implémentation de référence propose un seul modèle de rendu pour les composants qui propose de générer de l'HTML.

Ce modèle favorise la séparation entre l'état et le comportement d'un composant et sa représentation finale.

Le modèle de rendu permet de définir la représentation visuelle des composants. Chaque composant peut être rendu de plusieurs façons avec plusieurs modèles de rendu. Par exemple, un composant de type UICommand peut être rendu sous la forme d'un bouton ou d'un lien hypertexte. Dans cet exemple, le rendu est HTML mais il est possible d'utiliser d'autre système de rendu comme XML ou WML.

Le modèle de rendu met un oeuvre un plusieurs kits de rendus.

39.7.2. Utilisation de JSF dans une JSP

Pour une utilisation dans une JSP, l'implémentation de référence propose deux bibliothèques de tags personnalisés :

- core : cette bibliothèque contient des fonctionnalités de bases ne générant aucun rendu. L'utilisation de cette bibliothèque est obligatoire car elle contient notamment l'élément view
- html : cette bibliothèque se charge des composants avec un rendu en HTML

Pour utiliser ces deux bibliothèques, il est nécessaire d'utiliser une directive taglib pour chacune d'elle au début de page jsp.

Exemple :

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

Le préfix est libre mais par convention ce sont ceux fournis dans l'exemple qui sont utilisés.

Le tag <view> est obligatoire dans toutes pages utilisant JSF. Cet élément va contenir l'état de l'arborescence des composants de la page si l'application est configurée pour stocker l'état sur le client.

Le tag <form> génère un tag HTML form qui définit un formulaire.

Exemple :

```
<html>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<head>
  <title>Application de tests avec JSF</title>
</head>
<body>
  <h:form>
  ...
  </h:form>
</body>
</f:view>
</html>
```

39.8. La bibliothèque de tags Core

Cette bibliothèque est composée de 18 tags.

Tag	Rôle
actionListener	ajouter un listener pour une action sur composant
attribute	ajouter un attribut à un composant
convertDateTime	ajouter un convertisseur de type DateTime à un composant
convertNumber	ajouter un convertisseur de type numérique à un composant
facet	définit un élément particulier d'un composant
loadBundle	charger un fichier contenant les chaînes de caractères d'une locale dans une collection de type Map
param	ajouter un paramètre à un composant
selectitem	définir l'élément sélectionné dans un composant permettant de faire un choix
selectitems	définir les éléments sélectionnés dans un composant permettant de faire un choix
subview	définir une sous vue
verbatim	ajouter un texte brut à la vue
view	définir une vue
validator	ajouter un valideur à un composant
validateDoubleRange	ajouter un valideur de type « plage de valeurs réelles » à un composant
validateLength	ajouter un valideur de type « taille de la valeur » à un composant
validateLongRange	ajouter un valideur de type « plage de valeurs entières » à un composant
valueChangeListener	ajouter un listener pour un changement de valeur sur un composant

La plupart de ces tags permettent d'ajouter des objets à un composant. Leur utilisation sera détaillée tout au long de ce chapitre.

39.8.1. Le tag <selectItem>

Ce tag représente un élément dans un composant qui peut en contenir plusieurs.

Les attributs de base sont les suivants :

Attribut	Rôle
itemValue	contient la valeur de l'élément
itemLabel	contient le libellé de l'élément
itemDescription	contient une description de l'élément (utilisé uniquement par les outils de développement)
itemDisabled	contient l'état de l'élément
binding	contient le nom d'une méthode qui renvoie un objet de type javax.faces.model.SelectItem
id	contient l'identifiant du composant
value	contient une expression qui désigne un objet de type javax.faces.model.SelectItem

Exemple :

```
<f:selectItem value="#{test.elementSelectionne}"/>
```

L'attribut value attend en paramètre une expression qui désigne une méthode qui renvoie un objet de type SelectItem qui encapsule l'objet de la liste qui sera sélectionné.

Exemple :

```
...
public SelectItem getElementSelectionne() {
  return new SelectItem("Element 1");
}
...
```

La classe SelectItem possède quatre constructeurs qui permettent de définir les différentes propriétés qui composent l'élément.

39.8.2. Le tag <selectItems>

Ce tag représente une collection d'éléments dans un composant qui peut en contenir plusieurs.

Ce tag est particulièrement utile car il évite d'utiliser autant de tag selectItem que d'éléments à définir.

Exemple :

```
...
<h:selectOneRadio>
  <f:selectItems value="#{test.listeElements}"/>
</h:selectOneRadio>
...
```

La collection d'objets de type SelectItem peut être soit une collection soit un tableau.

Exemple : avec un tableau d'objects de type SelectItem

```
package com.jmd.test.jsf;

import javax.faces.model.SelectItem;

public class TestBean {

  private SelectItem[] elements = {
    new SelectItem(new Integer(1), "Element 1"),
    new SelectItem(new Integer(2), "Element 2"),
    new SelectItem(new Integer(3), "Element 3"),
```

```
    new SelectItem(new Integer(4), "Element 4"),
  };

  public SelectItem[] getListeElements() {
    return elements;
  }

...
}
```

La collection peut être de type Map : dans ce cas le framework associe la clé de chaque occurrence à la propriété itemValue et la valeur à la propriété itemLabel

Exemple :

```
package com.jmd.test.jsf;

import java.util.HashMap;
import java.util.Map;

import javax.faces.model.SelectItem;

public class TestBean {

  private Map elements = null;

  public Map getListeElements() {
    if (elements == null) {
      elements = new HashMap();
      elements.put("Element 1", new Integer(1));
      elements.put("Element 2", new Integer(2));
      elements.put("Element 3", new Integer(3));
      elements.put("Element 4", new Integer(4));
    }
    return elements;
  }

  public SelectItem getElementSelectionne() {
    return new SelectItem("Element 1");
  }

  ...
}
```

39.8.3. Le tag <verbatim>

Ce tag permet d'insérer du texte dans la vue.

Son utilisation est obligatoire dans le corps des tags JSF pour insérer autre chose qu'un tag JSF. Par exemple, pour insérer un tag HTML dans le corps d'un tag JSF, il est obligatoire d'utiliser le tag <verbatim>.

Les tags suivants peuvent avoir un corps : commandLink, outputLink, panelGroup, panelGrid et dataTable.

Exemple :

```
<h:outputLink value="http://java.sun.com" title="Java">
  <f:verbatim>
    Site Java de Sun
  </f:verbatim>
</h:outputLink>
```

Il est possible d'utiliser le tag <outputText> à la place du tag <verbatim>.

39.8.4. Le tag <attribute>

Ce tag permet de fournir un attribut quelconque à un composant puisque chaque composant peut stocker des attributs arbitraires.

Ce tag possède deux attributs :

Attribut	Rôle
Name	nom de l'attribut
Value	valeur de l'attribut

Dans le code d'un composant, il est possible d'utiliser la méthode getAttributes() pour obtenir une collection de type Map des attributs du composant.

Ceci permet de fournir un mécanisme souple pour fournir des paramètres sans être obligé de créer un nouveau composant ou de modifier un composant existant en lui ajoutant un ou plusieurs attributs.

39.8.5. Le tag <facet>

Ce tag permet de définir des éléments particuliers d'un composant.

Il est par exemple utilisé pour définir les lignes d'en−tête et de pied de page des tableaux.

Ce tag possède plusieurs attributs :

Attribut	Rôle
Name	Permet de préciser le type de l'élément généré par le tag Les valeurs possibles sont header et footer

Exemple :

```
<h:column>
  <f:facet name="header">
    <h:outputText value="Nom" />
  </f:facet>
  <h:outputText value="#{personne.nom}"/>
</h:column>
```

39.9. La bibliothèque de tags Html

Cette bibliothèque est composée de 25 tags qui permettent la réalisation de l'interface graphique de l'application.

Tag	Rôle
form	le tag <form> HTML
commandButton	un bouton
commandLink	un lien qui agit comme un bouton
graphicImage	une image

inputHidden	une valeur non affichée
inputSecret	une zone de saisie de texte mono ligne dont la valeur est non lisible
inputText	une zone de saisie de texte mono ligne
inputTextarea	une zone de saisie de texte multi−lignes
outputLink	un lien
outputFormat	du texte affiché avec des valeurs fournies en paramètre
outputText	du texte affiché
panelGrid	un tableau
panelGroup	un panneau permettant de regrouper plusieurs composants
selectBooleanCheckbox	une case à cocher
selectManyCheckbox	un ensemble de cases à cocher
selectManyListbox	une liste déroulante où plusieurs éléments sont sélectionnables
selectManyMenu	un menu où plusieurs éléments sont sélectionnables
selectOneListbox	une liste déroulante où un seul élément est sélectionnable
selectOneMenu	un menu où un seul élément est sélectionnable
selectOneRadio	un ensemble de boutons radio
dataTable	une grille proposant des fonctionnalités avancées
column	une colonne d'une grille
message	le message d'erreur lié à un composant
messages	les messages d'erreur liés à tous les composants

39.9.1. Les attributs communs

Ces tags possèdent des attributs communs pouvant être regroupés en trois catégories :

- les attributs de base
- les attributs liés à HTML
- les attributs liés à Javascript

Chaque tag utilise ou non chacun de ces attributs.

Les attributs de base sont les suivants :

Attribut	Rôle
id	contient l'identifiant du composant
binding	permet l'association avec un backing bean
rendered	contient un booléen qui indique si le composant doit être affiché
styleClass	contient le nom d'une classe CSS à appliquer au composant
value	contient la valeur du composant
valueChangeListener	permet l'association à une méthode qui va traiter les changements de valeurs
converter	contient une classe de conversion des données de chaîne de caractères en objet et vice et versa

validator	contient une classe de validation des données
required	contient un booléen qui indique si une valeur doit obligatoirement être saisie

L'attribut id est très important car il permet d'avoir accès :

- au tag dans le code de la vue par d'autres tags
  <h:inputText id="nom" required="true"/>

<h:message for="nom"/>

- au tag dans le code Javascript de la vue
- dans le code Java des objets métiers.
  UIComponent component = event.getComponent().findComponent("nomComposant");

L'attribut binding permet d'associer le composant avec un champ d'une classe de type bean. Un tel bean est nommé backing bean dans une application JSF.

Exemple :

```
...
<h:inputText value="#{login.nom}" id="nom" required="true" binding="#{login.inputTextNom}"/>
..

...
import javax.faces.component.UIComponent;

public class LoginBean {
  private String nom;

  private UIComponent inputTextNom;

  public UIComponent getInputTextNom() {
    return inputTextNom;
  }

  public void setInputTextNom(UIComponent inputTextNom) {
    this.inputTextNom = inputTextNom;
  }
...
```

L'attribut value permet de préciser la valeur d'un tag. Cette valeur peut être fournie sous deux formes :

- en dur dans le code :
  <h:outputText value="Bonjour"/>
- en utilisant une expression de liaison de données :
  <h:inputText value="#{login.nom}"/>

L'attribut converter permet de préciser une classe qui va convertir la valeur d'un objet en chaîne de caractères et vice et versa. L'utilisation de cet attribut est détaillée dans une des sections suivante.

L'attribut validator permet de préciser une classe qui va réaliser des contrôles de validation sur la valeur saisie. L'utilisation de cet attribut est détaillée dans une des sections suivante.

L'attribut styleClass permet de préciser le nom d'un style défini dans une feuille de style CSS qui sera appliqué au composant.

Exemple : le fichier monstyle.css

```
.titre {
color:red;
}
```

Dans la vue, il faut inclure la feuille de style dans la partie en−tête de la page HTML.

Exemple :

```
...
<link href="monstyle.css" rel="stylesheet" type="text/css"/>
...
<h:outputText value="#{msg.login_titre}" styleClass=”titre”/>
...
```

L'attribut renderer permet de préciser si le composant sera affiché ou non dans la vue. La valeur de l'attribut peut être obtenue dynamiquement par l'utilisation du langage d'expression.

Exemple :

```
<h:panelGrid rendered='#{listepersonnes.nbOccurences gt 0}'/>
```

Les principaux attributs liés à HTML sont les suivants :

Attribut	Rôle
accesskey	contient le raccourci clavier pour donner le focus au composant
alt	contient le texte alternatif pour les composants non textuels
border	contient la taille de la bordure en pixel
disabled	permet de désactiver le composant
maxlength	contient le nombre maximum de caractères saisis
readonly	permet de rendre une zone de saisie en lecture seule
rows	contient nombre de lignes visibles pour zone de saisie multi−ligne
shape	contient la définition d'une région
size	contient la taille de la zone de saisie
style	contient le style CSS à utiliser
target	contient le nom de la frame cible pour l'affichage de la page
title	contient le titre du composant généralement transformé en une bulle d'aide
width	contient la taille du composant

Le rôle de la plupart de ces tags est identique à leurs homologues définis dans HTML 4.0.

L'attribut style permet de définir un style CSS qui sera appliqué au composant. Cet attribut contient directement la définition du style à la différence de l'attribut styleClass qui contient le nom d'une classe CSS définit dans une feuille de style. Il est préférable d'utiliser l'attribut styleClass plutôt que l'attribut style afin de faciliter la maintenance de la charte graphique.

Exemple :

```
<h:outputText value="#{login.nom}" style="color:red;"/>
```

Les attributs liés à Javascript sont :

Attribut	Rôle
onblur	perte du focus

onchange	changement de la valeur
onclick	clic du bouton de la souris sur le composant
ondblclick	double−clic du bouton de la souris sur le composant
onfocus	réception du focus
onkeydown	une touche est enfoncée
onkeypress	appui sur une touche
onkeyup	une touché est relachée
onmousedown	
onmousemove	déplacement du curseur de la souris sur le composant
onmouseout	déplacement hors du cuseur de la souris hors du composant
onmouseover	passage de la souris au dessus du composant
onmouseup	le bouton de la souris est relachée
onreset	réinitialisation du formulaire
onselect	sélection du texte dans une zone de saisie
onsubmit	soumission du formulaire

39.9.2. Le tag <form>

Ce tag représente un formulaire HTML.

Il possède les attributs suivants :

Attributes	Rôle
binding, id, rendered, styleClass	attributs communs de base
accept, acceptcharset, dir, enctype, lang, style, target, title	attributs communs liés à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onreset, onsubmit	attributs communs liés aux événements Javascript

Il est préférable de définir explicitement l'attribut id pour permettre son exploitation notamment dans le code Javascript, sinon un id est généré automatiquement.

Ceci est d'autant plus important que les id des composants intégrés dans le formulaire sont préfix par l'id du formulaire suivi du caractère deux points. Il faut tenir compte de ce point lors de l'utilisation de code Javascript faisant référence à un composant.

39.9.3. Les tags <inputText>, <inputTextarea>, <inputSecret>

Ces trois composants permettent de générer des composants pour la saisie de données.

Les attributs de ces tags sont les suivants :

Attributs	Rôle
cols	définir le nombre de colonne (pour le composant inputTextarea uniquement)
immediate	permettre de demander d'ignorer les étapes de validation des données
redisplay	permettre de réafficher le contenu lors du réaffichage de la page (pour le composant inputSecret uniquement)
required	rendre obligatoire la saisie d'une valeur
rows	définir le nombre de lignes affichées (pour le composant inputTextarea uniquement)
valueChangeListener	préciser une classe de type listener lors du changement de la valeur
binding, converter, id, rendered, required, styleClass, value, validator	attributs communs de base
accesskey, alt, dir, disabled,lang, maxlength, readonly, size, style, tabindex, title	attributs communs liés à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onselect	attributs communs lies aux événements Javascript

Exemple :

```
<html>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<head>
  <title>Saisie des données</title>
</head>
<body>
  <h:form>
    <h3>Saisie des données</h3>
    <p><h:inputText size="20" /></p>
    <p><h:inputTextarea rows="3" cols="20" /></p>
    <p><h:inputSecret size="20" /></p>
  </h:form>
</body>
</f:view>
</html>
```

Résultat

39.9.4. Le tag <ouputText> et <outputFormat>

Ces deux tags permettent d'insérer une valeur sous la forme d'une chaîne de caractères dans la vue. Par défaut, ils ne génèrent pas de tag HTML mais insèrent simplement la valeur dans la vue sauf si un style CSS est précisé avec l'attribut style ou styleClass. Dans ce cas, la valeur est contenue un tag HTML .

Les attributs de ces deux tags sont :

Attributs	Rôle
escape	booléen qui précise si certains caractères de la valeur seront encodé ou non. La valeur par défaut est false.
binding, converter, id, rendered, styleClass, value	attributs communs de base
style, title	attributs communs lies à HTML

L'attribut escape est particulièrement utile pour encoder certains caractères spéciaux avec leur code HTML correspondant.

Exemple :

```
<h:outputText escape="true" value="Nombre d'occurences > 200"/>
```

Le tag outputText peut être utilisé pour générer du code HTML en valorisant l'attribut escape à false.

Exemple :

```
<p><h:outputText escape="false" value="<H2>Saisie des données</H2>"/></p>

<p><h:outputText escape="true" value="<H2>Saisie des données</H2>"/></p>
```

Résultat :

Saisie des données

<H2>Saisie des données</H2>

Le tag outputFormat permet de formater une chaîne de caractères avec des valeurs fournies en paramètres.

Exemple :

```
<p>
  <h:outputFormat value="La valeur doit être entre {0} et {1}.">
    <f:param value="1"/>
    <f:param value="9"/>
  </h:outputFormat>
</p>
```

Résultat

La valeur doit être entre 1 et 9.

Ce composant utilise la classe java.text.MessageFormat pour formater le message. L'attribut value doit donc contenir une chaîne de caractères utilisable par cette classe.

Le tag <param> permet de fournir la valeur de chacun des paramètres.

39.9.5. Le tag <graphicImage>

Ce composant représente une image : il génère un tag HTML <img>.

Les attributs de ce tag sont les suivants :

Attributs	Rôle
binding, id, rendered, styleClass, value	Attributs communs de base
alt, dir, height, ismap, lang, longdesc, style, title, url, usemap, width	Attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup	Attributs communs lies aux événements Javascript

Les attributs value et url peuvent préciser l'url de l'image.

Exemple :

```
<p><h:graphicImage value="/images/erreur.jpg" /></p>
<p><h:graphicImage url="/images/warning.jpg" /></p>
```

Résultat

39.9.6. Le tag <inputHidden>

Ce composant représente un champ caché dans un formulaire.

Les attributs sont les suivants :

Attributs	Rôle
binding, converter, id, immediate, required, validator, value, valueChangeListener	attributs communs de base

Exemple :

```
<h:inputHidden value="#{login.nom}" />
```

Résultat dans le code HTML

```
...
          <input type="hidden" name="_id0:_id12" value="test" />
...
```

39.9.7. Le tag <commandButton> et <commandLink>

Ces composants représentent respectivement un bouton de formulaire et un lien qui déclenche une action. L'action demandée sera traitée par le framework JSF.

Les attributs sont les suivants :

Attributs	Rôle
action	peut être une chaîne de caractères ou une méthode qui renvoie une chaîne de caractères qui sera traitée par le navigation handler.
actionListener	précise une méthode possédant une signature void nomMethode(ActionEvent) qui sera exécutée lors d'un clic
image	url, tenant compte du contexte de l'application, de l'image qui sera utilisée à la place du bouton (uniquement pour le tag commandButton)
type	type de bouton généré : button, submit, reset (uniquement pour le tag commandButton)
value	le texte affiché par le bouton ou le lien
accesskey, alt, binding, id, lang, rendered, styleClass	attributs communs de base
coords (uniquement pour le tag commandLink), dir, disabled, hreflang (uniquement pour le tag commandLink), lang, readonly, rel (uniquement pour le tag commandLink), rev (uniquement pour le tag commandLink), shape (uniquement pour le tag commandLink), style, tabindex, target (uniquement pour le tag commandLink), title	attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	attributs communs lies aux événements Javascript

Il est possible d'insérer dans le corps du tag <commandLink> d'autres composants qui feront partie intégrante du lien comme par exemple du texte ou une image.

Exemple :

```
<p>
  <h:commandLink>
    <h:outputText value="Valider"/>
  </h:commandLink>
</p>
<p>
  <h:commandLink>
    <h:graphicImage value="/images/oeil.jpg"/>
  </h:commandLink>
</p>
```

Résultat

Il est aussi possible de fournir un ou plusieurs paramètres qui seront envoyés dans la requête en utilisant le tag <param> dans le corps du tag

Exemple :

```
<h:commandLink>
  <h:outputText value="Selectionner"/>
  <f:param name="id" value="1"/>
</h:commandLink>
```

Résultat dans le page HTML générée

```
<a href="#" onclick="document.forms['_id0']['_id0:_idcl'].value='_id0:_id15';
          document.forms['_id0'].submit(); return false;">
        mg src="/test_JSF/images/oeil.jpg" alt="" /></a>
```

Le tag <commandLink> génère dans la vue du code Javascript pour soumettre le formulaire lors d'un clic.

39.9.8. Le tag <ouputLink>

Ce composant représente un lien direct vers une ressource dont la demande ne sera pas traitée par le framework JSF.

Les attributs sont les suivants :

Attributs	Rôle
accesskey, binding, converter, id, lang, rendered, styleClass, value	attributs communs de base
charset, coords, dir, hreflang, lang, rel, rev, shape, style, tabindex, target, title, type	attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup	attributs communs lies aux événements Javascript

L'attribut value doit contenir l'url qui sera utilisée dans l'attribut href du lien HTML. Si le première caractère est un # (dièse) alors le lien pointe vers une ancre définie dans la même page.

Il est possible d'insérer dans le corps du tag ouputLink d'autres composants qui feront partie intégrante du lien.

Exemple :

```
<p>
  <h:outputLink value="http://java.sun.com">
    <h:graphicImage value="/images/java.jpg"/>
  </h:outputLink>
</p>
```

Résultat

Le code HTML généré dans la page est le suivant :

Attention, pour mettre du texte dans le corps du tag, il est nécessaire d'utiliser un tag verbatim ou outputText.

Exemple :

```
<p>
  <h:outputLink value="http://java.sun.com" title="Java">
    <f:verbatim>
       Site Java de Sun
    </f:verbatim>
  </h:outputLink>
</p>
```

39.9.9. Les tags <selectBooleanCheckbox> et <selectManyCheckbox>

Ces composants représentent respectivement une case à cocher et un ensemble de cases à cocher.

Les attributs sont les suivants :

Attributs	Rôle
disabledClass	classe CSS pour les éléments non sélectionnés (pour le tag selectManyCheckbox uniquement)
enabledClass	classe CSS pour les éléments sélectionnés (pour le tag selectManyCheckbox uniquement)
layout	préciser la disposition des éléments (pour le tag selectManyCheckbox uniquement)
binding, converter, id, immediate, styleClass, required, rendered, validator, value, valueChangeListener	attributs communs de base
accesskey, border, dir, disabled, lang, readonly, style, tabindex, title	attributs communs lies à HTML (border pour le tag selectManyCheckbox uniquement)
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	attributs communs lies aux événements Javascript

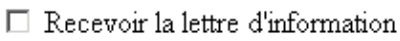
L'attribut layout permet de préciser la disposition des cases à cocher : lineDirection pour une disposition horizontale (c'est la valeur par défaut) et pageDirection pour une disposition verticale.

Le tag <selectBooleanCheckbox> dont la valeur peut être associée à une propriété booléenne d'un bean représente une case à cocher simple.

Exemple :

```
<h:selectBooleanCheckbox value="#{saisieOptions.recevoirLettre}">
</h:selectBooleanCheckbox> Recevoir la lettre d'information
```

Résultat :



Pour gérer l'état du composant, il faut utiliser l'attribut value en lui fournissant en valeur une propriété booléen d'un backing bean.

Exemple :

```
public class SaisieOptions {

  private boolean recevoirLettre;

  public void setRecevoirLettre(boolean valeur) {
    recevoirLettre = valeur;
  }

  public boolean getRecevoirLettre() {
    return recevoirLettre;
  }
...
```

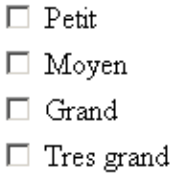
Le tag <selectManyCheckbox> représente un ensemble de cases à cocher. Dans cet ensemble, il est possible de sélectionner une ou plusieurs cases à cocher.

Chaque case à cocher est définie par un tag selectItem dans le corps du tag selectMenyCheckbox.

Exemple :

```
<h:selectManyCheckbox layout="pageDirection">
  <f:selectItem itemValue="petit" itemLabel="Petit" />
  <f:selectItem itemValue="moyen" itemLabel="Moyen" />
  <f:selectItem itemValue="grand" itemLabel="Grand" />
  <f:selectItem itemValue="tresgrand" itemLabel="Tres grand" />
</h:selectManyCheckbox>
```

Résultat :



Le rendu du composant est un tableau HTML dont chaque cellule contient une case à cocher encapsulée dans un tag HTML <label> :

Exemple :

```
<table>
  <tr>
    <td>
      <label><input name="_id0:_id1" value="petit" type="checkbox"> Petit</input></label></td>
  </tr>
  <tr>
    <td>
      <label><input name="_id0:_id1" value="moyen" type="checkbox"> Moyen</input></label></td>
  </tr>
  <tr>
    <td>
      <label><input name="_id0:_id1" value="grand" type="checkbox"> Grand</input></label></td>
  </tr>
```

```
  <tr>
    <td>
      <label><input name="_id0:_id1" value="tresgrand" type="checkbox"> Tres grand</input>
      </label></td>
  </tr>
</table>
```

39.9.10. Le tag <selectOneRadio>

Ce composant représente un ensemble de boutons radio dont un seul peut être sélectionné.

Les attributs sont les suivants :

Attributs	Rôle
disabledClass	classe CSS pour les éléments non sélectionnés
enabledClass	classe CSS pour les éléments sélectionnés
layout	préciser la disposition des éléments
binding, converter, id, immediate, styleClass, required, rendered, validator, value, valueChangeListener	Attributs communs de base
accesskey, border, dir, disabled, lang, readonly, style, tabindex, title	Attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	Attributs communs lies aux événements Javascript

Les éléments peuvent être precisés un par un avec le tag <selectItem>.

Exemple :

```
<h:selectOneRadio layout="pageDirection">
  <f:selectItem itemValue="petit" itemLabel="Petit" />
  <f:selectItem itemValue="moyen" itemLabel="Moyen" />
  <f:selectItem itemValue="grand" itemLabel="Grand" />
  <f:selectItem itemValue="tresgrand" itemLabel="Tres grand" />
</h:selectOneRadio>
```

Résultat :

- ○ Petit
- ○ Moyen
- ○ Grand
- ○ Tres grand

Le rendu du composant est un tableau HTML dont chaque cellule contient un bouton radio encapsulé dans un tag HTML <label> :

Exemple :

```
<table>
  <tr>
```

```
    <td>
      <label><input type="radio" name="_id0:_id1" value="petit"> Petit</input></label></td>
  </tr>
  <tr>
    <td>
      <label><input type="radio" name="_id0:_id1" value="moyen"> Moyen</input></label></td>
  </tr>
  <tr>
    <td>
      <label><input type="radio" name="_id0:_id1" value="grand"> Grand</input></label></td>
  </tr>
  <tr>
    <td>
      <label><input type="radio" name="_id0:_id1" value="tresgrand"> Tres grand</input>
      </label></td>
  </tr>
</table>
```

Les éléments peuvent être precisés sous la forme d'un tableau de type SelecItem avec le tag <selectItems>.

Exemple :

```
<h:selectOneRadio value="#{saisieOptions.taille}" layout="pageDirection" id="taille">
<f:selectItems value="#{saisieOptions.tailleItems}"/>
</h:selectOneRadio>
```

Dans ce cas, le bean doit contenir au moins deux méthodes : getTaille() pour renvoyer la valeur de l'élément sélectionner et getTailleItems() qui renvoie un tableau d'objets de type SelectItems contenant les éléments.

Exemple :

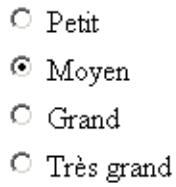
```
package com.jmd.test.jsf;

import javax.faces.model.*;

public class SaisieOptions {

  private Integer taille = null;

  private SelectItem[] tailleItems = {
    new SelectItem(new Integer(1), "Petit"),
    new SelectItem(new Integer(2), "Moyen"),
    new SelectItem(new Integer(3), "Grand"),
    new SelectItem(new Integer(4), "Très grand") };

  public SaisieOptions() {
    taille = new Integer(2);
  }

  public Integer getTaille() {
    return taille;
  }

  public void setTaille(Integer newValue) {
    taille = newValue;
  }

  public SelectItem[] getTailleItems() {
    return tailleItems;
  }
}
```

Le bean doit être déclaré dans le fichier faces−config.xml

Exemple :

```
<managed-bean>
  <managed-bean-name>saisieOptions</managed-bean-name>
  <managed-bean-class>com.jmd.test.jsf.SaisieOptions</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Résultat :



39.9.11. Le tag <selectOneListbox>

Ce composant représente une liste d'éléments dont un seul peut être sélectionné

Les attributs sont les suivants :

Attributs	Rôle
binding, converter, id, immediate, styleClass, required, rendered, validator, value, valueChangeListener	attributs communs de base
accesskey, dir, disabled, lang, readonly, style, size, tabindex, title	attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	attributs communs lies aux événements Javascript

L'attribut size permet de préciser le nombre d'éléments de la liste affiché.

Exemple :

```
<h:selectOneListbox value="#{saisieOptions.taille}">
  <f:selectItems value="#{saisieOptions.tailleItems}"/>
</h:selectOneListbox>
```

Résultat :

39.9.12. Le tag <selectManyListbox>

Ce composant représente une liste d'éléments dont plusieurs peuvent être sélectionnés.

Les attributs sont les suivants :

Attributs	Rôle

binding, converter, id, immediate, styleClass, required, rendered, validator, value, valueChangeListener	attributs communs de base
accesskey, dir, disabled, lang, readonly, style, size, tabindex, title	attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	attributs communs lies aux événements Javascript

Exemple :

```
<h:selectManyListbox value="#{saisieOptions.legumes}">
  <f:selectItems value="#{saisieOptions.legumesItems}"/>
</h:selectManyListbox>
```

La liste des éléments sélectionnés doit pouvoir contenir zero ou plusieurs valeurs sous la forme d'un tableau ou d'une liste.

Exemple :

```
package com.jmd.test.jsf;

import javax.faces.model.*;

public class SaisieOptions {

  private String[] legumes = {
    "navets", "choux" };
  private SelectItem[] legumesItems = {
    new SelectItem("epinards", "Epinards"),
    new SelectItem("poireaux", "Poireaux"),
    new SelectItem("navets", "Navets"),
    new SelectItem("flageolets", "Flageolets"),
    new SelectItem("choux", "Choux"),
    new SelectItem("aubergines", "Aubergines") };

  public SaisieOptions() {
  }

  public String[] getLegumes() {
    return legumes;
  }

  public SelectItem[] getLegumesItems() {
    return legumesItems;
  }
}
```

Résultat :

Il est possible d'utiliser un objet de type List à la place des tableaux.

Exemple :

```
package com.jmd.test.jsf;

import javax.faces.model.*;
import java.util.*;
```

```
public class SaisieOptions {

  private List legumes = null;

  private List legumesItems = null;

  public List getLegumesItems() {
    if (legumesItems == null) {
      legumesItems = new ArrayList();
      legumesItems.add(new SelectItem("epinards", "Epinards"));
      legumesItems.add(new SelectItem("poireaux", "Poireaux"));
      legumesItems.add(new SelectItem("navets", "Navets"));
      legumesItems.add(new SelectItem("flageolets", "Flageolets"));
      legumesItems.add(new SelectItem("choux", "Choux"));
      legumesItems.add(new SelectItem("aubergines", "Aubergines"));
    }
    return legumesItems;
  }

  public List getLegumes() {
    return legumes;
  }

  public void setLegumes(List newValue) {
    legumes = newValue;
  }

  public SaisieOptions() {
    legumes = new ArrayList();
    legumes.add("navets");
    legumes.add("choux");
  }
}
```

39.9.13. Le tag <selectOneMenu>

Ce composant représente une liste déroulante dont un seul élément peut être sélectionné.

Les attributs sont les suivants :

Attributs	Rôle
binding, converter, id, immediate, styleClass, required, rendered, validator, value, valueChangeListener	attributs communs de base
accesskey, dir, disabled, lang, readonly, style, tabindex, title	attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	attributs communs lies aux événements Javascript

Exemple :

```
<h:selectOneMenu value="#{saisieOptions.taille}">
<f:selectItems value="#{saisieOptions.tailleItems}"/>
</h:selectOneMenu>
```

Résultat :

Exemple : Le code HTML généré

```
<select name="_id0:_id6" size="1"> <option value="1">Petit</option>
  <option value="2" selected="selected">Moyen</option>
  <option value="3">Grand</option>
  <option value="4">Tr&egrave;s grand</option>
</select>
```

39.9.14. Le tag <selectManyMenu>

Ce composant représente une liste d'éléments dont le rendu HTML est un tag select avec une seule option visible.

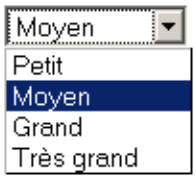
Les attributs sont les suivants :

Attributs	Rôle
binding, converter, id, immediate, styleClass, required, rendered, validator, value, valueChangeListener	Attributs communs de base
accesskey, dir, disabled, lang, readonly, style, tabindex, title	Attributs communs lies à HTML
onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect	Attributs communs lies aux événements Javascript

Exemple :

```
<h:selectManyMenu value="#{saisieOptions.legumes}">
<f:selectItems value="#{saisieOptions.legumesItems}"/>
</h:selectManyMenu>
```

Résultat :



39.9.15. Les tags <message> et <messages>

Des messages peuvent être émis lors de traitements. Ils sont stockés dans le contexte de l'application JSF pour être restitués dans la vue. Ils permettent notamment de fournir des messages d'erreur aux utilisateurs.

JSF défini quatre types de message :

- Information
- Warning
- Error
- Fatal

Chaque message possède un résumé et un descriptif.

Le tag <messages> permet d'afficher tous les messages stockés dans le contexte de l'application JSF.

Le tag message permet d'afficher un seul message, le dernier ajouté, pour un composant donné.

Les attributs sont les suivants :

Attributs	Rôle
errorClass	nom d'une classe CSS pour un message de type error
errorStyle	style CSS pour un message de type error
fatalClass	nom d'une classe CSS pour un message de type fatal
fatalStyle	style CSS pour un message de type fatal
globalOnly	booléen qui permet de n'affiche que les messages qui ne sont pas associés à un composant. Par défaut, False (uniquement pour le tag messages)
infoClass	nom d'une classe CSS pour un message de type Information
infoStyle	style CSS pour un message de type Information
Layout	format de la liste de messages : list ou table (uniquement pour le composant messages)
showDetail	booléen qui précise si la description des messages est affichée ou non. Par défaut, false pour le tag message et true pour le tag messages
showSummary	booléen qui précise si le résumé des messages est affichée ou non. Par défaut, true pour le tag message et false pour le tag messages
Tooltip	booléen qui précise si la description est afficher sous la forme d'une bulle d'aide
warnClass	nom d'une classe CSS pour un message de type Warning
warnStyle	Style CSS pour un message de type Warning
For	l'identifiant du composant pour lequel le message doit être affiché
binding, id, rendered, styleClass	attributs communs de base
style, title	attributs communs lies à HTML

39.9.16. Le tag <panelGroup>

Ce composant permet de regrouper plusieurs composants.

Les attributs sont les suivants :

Attributs	Rôle
binding, id, rendered, styleClass	attributs communs de base
Style	attributs communs lies à HTML

Exemple :

```
<td bgcolor='#DDDDDD'>
  <h:panelGroup>
    <h:inputText value="#{login.nom}" id="nom" required="true"/>
    <h:message for="nom"/>
  </h:panelGroup>
</td>
```

Résultat :

39.9.17. Le tag <panelGrid>

Ce composant représente un tableau HTML.

Les attributs sont les suivants :

Attributs	Rôle
Bgcolor	couleur de fond du tableau
Border	taille de la bordure du tableau
Cellpadding	espacement intérieur de chaque cellule
Cellspacing	espacement extérieur de chaque cellule
columnClasses	nom de classes CSS pour les colonnes. Il est possible de préciser plusieurs noms de classe qui seront utilisées sur chaque colonne
Columns	nombre de colonne du tableau
footerClass	nom de la classe CSS pour le pied du tableau
Frame	précise les règles pour le contour du tableau. Les valeurs possibles sont : none, above, below, hsides, vsides, lhs, rhs, box, border
headerClass	nom de la classe CSS pour l'en−tête du tableau
rowClasses	nom de classes CSS pour les lignes. Il est possible de préciser deux noms de classe séparés par une virgule qui seront utilisés alternativement sur chaque ligne
Rules	précise les règles de dessin des lignes entre les cellules. Les valeurs possibles sont : groups, rows, columns, all
Summary	résumé du tableau
binding, id, rendered, styleClass, value	attributs communs de base
dir, lang, style, title, width	attributs communs lies à HTML
onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup	attributs communs lies aux événements Javascript

Par défaut chaque composant est inséré les uns à la suite des autres dans les cellules en partant de la gauche vers la droite en passant à ligne suivante dès que nécessaire.

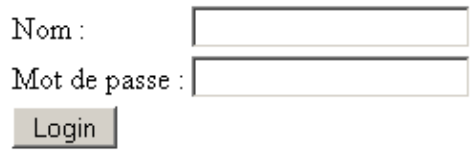
Il n'est possible de mettre qu'un seul composant par cellule. Ainsi pour mettre plusieurs composants dans une cellule, il faut les regrouper dans un tag panelGroup.

Exemple :

```
<h:panelGrid columns="2">
  <h:outputText value="Nom : " />
  <h:panelGroup>
    <h:inputText value="#{login.nom}" id="nom" required="true"/>
    <h:message for="nom"/>
  </h:panelGroup>
  <h:outputText value="Mot de passe :" />
  <h:inputSecret value="#{login.mdp}"/>
```

```
  <h:commandButton value="Login" action="login"/>
</h:panelGrid>
```

Résultat :



Le code HTML généré est le suivant :

Exemple : Le code HTML généré

```
...
<table>

  <tbody>
    <tr>
      <td>Nom : </td>
      <td><input id="_id0:nom" type="text" name="_id0:nom" /></td>
    </tr>
    <tr>
      <td>Mot de passe :</td>
      <td><input type="password" name="_id0:_id6" value="" /></td>
    </tr>
    <tr>
      <td><input type="submit" name="_id0:_id7" value="Login" /></td>
    </tr>
  </tbody>
</table>
...
```

39.9.18. Le tag <dataTable>

Ce composant représente un tableau HTML dans lequel des données vont pouvoir être automatiquement présentées. Ce composant est sûrement le plus riche en fonctionnalité et donc le plus complexe des composants fournis en standard.

Les attributs sont les suivants :

Attributs	Rôle
Bgcolor	couleur de fond du tableau
Border	taille de la bordure du tableau
Cellpadding	espacement intérieur de chaque cellule
Cellspacing	espacement extérieur de chaque cellule
columnClasses	nom de classes CSS pour les colonnes. Il est possible de préciser plusieurs noms de classe qui seront utilisées sur chaque colonne
First	index de la première occurrences des données qui sera affichée dans le tableau
footerClass	nom de la classe CSS pour le pied du tableau
Frame	précise les règles pour le contour du tableau. Les valeurs possibles sont : none, above, below, hsides, vsides, lhs, rhs,

	box, border
headerClass	nom de la classe CSS pour l'en−tête du tableau
rowClasses	nom de classes CSS pour les lignes. Il est possible de préciser deux noms de classe qui seront utilisées alternativement sur chaque ligne
Rules	précise les règles de dessin des lignes entre les cellules. Les valeurs possibles sont : groups, rows, columns, all
Summary	résumé du tableau
Var	nom de la variable qui va contenir l'occurrence en cours de traitement lors du parcours des données
binding, id, rendered, styleClass, value	attributs communs de base
dir, lang, style, title, width	attributs communs lies à HTML
onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup	attributs communs lies aux événements Javascript

Le tag <dataTable> parcours les données et pour chaque occurrence, il créé une ligne dans le tableau.

L'attribut value représente une expression qui précise les données à utiliser. Ces données peuvent être sous la forme :

- d'un tableau
- d'un objet de type java.util.List
- d'un objet de type java.sql.ResultSet
- d'un objet de type javax.servlet.jsp.jstl.sql.Result
- d'un objet de type javax.faces.model.DataModel

Pour chaque élément encapsulés dans les données, le tag dataTable créé une nouvelle ligne.

Quelque soit le type qui encapsule les données, le composant dataTable va les mapper dans un objet de type DataModel. C'est cet objet que le composant va utiliser comme source de données. JSF définit 5 classes qui héritent de la classe DataModel : ArrayDataModel, ListDataModel, ResultDataModel, ResultSetDataModel et ScalarDataModel.

La méthode getWrappedObject() permet d'obtenir la source de données fournie en paramètre de l'attribut value.

L'attribut item permet de préciser le nom d'une variable qui va contenir les données d'une occurrence.

Chaque colonne est définie grâce à un tag <column>.

Exemple :

```
<h:dataTable value="#{listePersonnes.personneItems}" var="personne" cellspacing="4">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Nom" />
    </f:facet>
    <h:outputText value="#{personne.nom}"/>
  </h:column>

  <h:column>
    <f:facet name="header">
      <h:outputText value="Prenom" />
    </f:facet>
    <h:outputText value="#{personne.prenom}" />
  </h:column>

  <h:column>
    <f:facet name="header">
      <h:outputText value="Date de naissance" />
    </f:facet>
```

```
    <h:outputText value="#{personne.datenaiss}"/>
  </h:column>

  <h:column>
    <f:facet name="header">
      <h:outputText value="Poids" />
    </f:facet>
    <h:outputText value="#{personne.poids}"/>
  </h:column>

  <h:column>
    <f:facet name="header">
      <h:outputText value="Taille" />
    </f:facet>
    <h:outputText value="#{personne.taille}"/>
  </h:column>

</h:dataTable>
```

L'en−tête et le pied du tableau sont précisés avec un tag <facet> pour chacun dans chaque tag <column>

Dans l'exemple précédent l'instance listePersonnes est une classe dont le code est le suivant :

Exemple :

```
package com.jmd.test.jsf;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;

public class PersonnesBean {

  private List PersonneItems = null;

  public List getPersonneItems() {
    if (PersonneItems == null) {
      PersonneItems = new ArrayList();
      PersonneItems.add(new Personne("Nom1", "Prenom1",
        new GregorianCalendar(1967, Calendar.OCTOBER, 22).getTime(),10,1.10f));
      PersonneItems.add(new Personne("Nom2", "Prenom2",
        new GregorianCalendar(1972, Calendar.MARCH, 10).getTime(),20,1.20f));
      PersonneItems.add(new Personne("Nom3", "Prenom3",
        new GregorianCalendar(1944, Calendar.NOVEMBER, 4).getTime(),30,1.30f));
      PersonneItems.add(new Personne("Nom4", "Prenom4",
        new GregorianCalendar(1958, Calendar.JULY, 19).getTime(),40,1.40f));
      PersonneItems.add(new Personne("Nom5", "Prenom5",
        new GregorianCalendar(1934, Calendar.JANUARY, 6).getTime(),50,1.50f));
      PersonneItems.add(new Personne("Nom6", "Prenom6",
        new GregorianCalendar(1989, Calendar.DECEMBER, 12).getTime(),60,1.60f));
    }
    return PersonneItems;
  }

}
```

La méthode getPersonneItems() renvoie une collection d'objets de type Personne.

La classe Personne encapsule simplement les données d'une personne.

Exemple :

```
package com.jmd.test.jsf;

import java.util.Date;

public class Personne {
```

```
  private String nom;
  private String prenom;
  private Date datenaiss;
  private int poids;
  private float taille;
  private boolean supprime;

  public Personne(String nom, String prenom, Date datenaiss, int poids,
    float taille) {
    super();
    this.nom = nom;
    this.prenom = prenom;
    this.datenaiss = datenaiss;
    this.poids = poids;
    this.taille = taille;
    this.supprime = false;
  }

  public boolean isSupprime() {
    return supprime;
  }

  public void setSupprime(boolean supprimer) {
    supprime = supprimer;
  }

  public Date getDatenaiss() {
    return datenaiss;
  }

  public void setDatenaiss(Date datenaiss) {
    this.datenaiss = datenaiss;
  }

  public String getNom() {
    return nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
  }

  public int getPoids() {
    return poids;
  }

  public void setPoids(int poids) {
    this.poids = poids;
  }

  public String getPrenom() {
    return prenom;
  }

  public void setPrenom(String prenom) {
    this.prenom = prenom;
  }

  public float getTaille() {
    return taille;
  }

  public void setTaille(float taille) {
    this.taille = taille;
  }
}
```

Il est très facile de préciser un style particulier pour des lignes paires et impaires.

Il suffit de définir les deux styles désirés.

Exemple dans la partie en-tête de la JSP :

```
<STYLE type="text/css">
<!--
.titre {
background-color:#000000;
color:#FFFFFF;
}
.paire {
background-color:#EFEFEF;
}
.impaire {
background-color:#CECECE;
}
-->
</STYLE>
```

Il suffit d'utiliser les attributs headerClass, footerClass, rowClasses ou columnClasses. Avec ces deux derniers attributs, il est possible de préciser plusieurs style séparés par une virgule pour définir le style de chacune des lignes de façons répétitives.

Exemple :

```
<h:dataTable value="#{listePersonnes.personneItems}" var="personne"
  cellspacing="4" width="60%" rowClasses="paire,impaire" headerClass="titre">
```

Résultat :

Nom	Prenom	Date de naissance	Poids	Taille
Nom1	Prenom1	22/10/1967	10	1.1
Nom2	Prenom2	10/03/1972	20	1.2
Nom3	Prenom3	04/11/1944	30	1.3
Nom4	Prenom4	19/07/1958	40	1.4
Nom5	Prenom5	06/01/1934	50	1.5
Nom6	Prenom6	12/12/1989	60	1.6

Les éléments du tableau peuvent par exemple être sélectionnés grace à une case à cocher pour permettre de réaliser des traitements sur les éléments marqués.

Il suffit de rajouter dans l'exemple précedent une colonne contenant une case et cocher et ajouter un bouton qui va réaliser les traitements sur les éléments cochés.

Exemple dans la JSP :

```
...
<h:form>
  <h1>Test</H1>
  <div align="center">
    <h:dataTable value="#{listePersonnes.personneItems}" var="personne"
      cellspacing="4" width="60%" rowClasses="paire,impaire" headerClass="titre">
...
    <h:column>
      <f:facet name="header">
        <h:outputText value="Sélection"/>
      </f:facet>
      <h:selectBooleanCheckbox value="#{personne.supprime}" />
    </h:column>

    </h:dataTable>
```

```
    <p>
      <h:commandButton value="Supprimer les sélectionnés"
        action="#{listePersonnes.supprimer}"/>
    </p>

  </div>
</h:form>
...
```

Il suffit alors d'ajouter les traitements dans la méthode supprimer() de la classe PersonnesBean qui sera appelée lors d'un clic sur le bouton « Supprimer les sélectionnés ».

Exemple :

```
public class PersonnesBean {

...

  public String supprimer() {
    Iterator iterator = personneItems.iterator();
    Personne pers=null;
    while (iterator.hasNext()) {
      pers = (Personne) iterator.next();
      System.out.println("nom="+pers.getNom()+" "+pers.isSupprime());
      // ajouter les traitements utiles
    }
    return null;
  }
}
```

Nom	Prenom	Date de naissance	Poids	Taille	Sélection
Nom1	Prenom1	22/10/1967	10	1.1	☐
Nom2	Prenom2	10/03/1972	20	1.2	☑
Nom3	Prenom3	04/11/1944	30	1.3	☐
Nom4	Prenom4	19/07/1958	40	1.4	☑
Nom5	Prenom5	06/01/1934	50	1.5	☐
Nom6	Prenom6	12/12/1989	60	1.6	☑

Supprimer les sélectionnés

Un clic sur le bouton « Supprimer les sélectionnés » affiche dans la console, la liste des éléments avec l'état de la case à cocher.

Exemple :

```
nom=Nom1 false
nom=Nom2 true
nom=Nom3 false
nom=Nom4 true
nom=Nom5 false
nom=Nom6 true
```

39.10. La gestion et le stockage des données

Les données sont stockées dans un ou plusieurs java bean qui encapsulent les différentes données des composants.

Ces données possèdent deux représentations :

- une contenue en interne par le modèle
- une pour leur présentation dans l'interface graphique (pour la saisie ou l'affichage)

Chaque objet de type Renderer possède une représentation par défaut des données. La transformation d'une représentation en une autre est assurée par des objets de type Converter. JSF fourni en standard plusieurs objets de type Converter mais il est aussi possible de développer ces propres objets.

39.11. La conversion des données

JSF propose en standard un mécanisme de conversion des données. Celui ci repose sur un ensemble de classes dont certaines sont fournis en standard pour des conversions de base. Il est possible de définir ces propres classes de conversion pour répondre à des besoins spécifiques.

Ces conversions sont nécessaires car toutes les données transmises et affichées le sont sous la forme de chaîne de caractères. Cependant, leur exploitation dans les traitements nécessite souvent qu'elles soient stockées dans un autre format pour être exploité : un exemple flagrant est une données de type date.

Toutes les données saisies par l'utilisateur sont envoyées dans la requête http sont sous la forme de chaînes de caractères. Chacune de ces valeurs est désignée par « request value » dans les spécifications de JSF.

Ces valeurs sont stockées dans leur composant respectif dans des champs désignés par « submitted value » dans les spécifications.

Ces valeurs sont ensuite éventuellement converties implicitement ou explicitement et sont stockées dans leur composant respectif dans des champs désignés par « local value ». Ces données sont ensuite éventuellement validées.

L'intérêt d'un tel procédé est de s'assurer que les données seront valides avant de pouvoir les utiliser dans les traitements. Si la conversion ou la validation échoue, les traitements du cyle de vie de la page sont arrêtés et la page est réaffichée pour permettre l'affichage de messages d'erreurs. Sinon la phase de mise à jour des données (« Update model values ») du modèle est exécutée.

Les spécifications JSF imposent l'implémentation des convertisseurs suivants : javax.faces.DateTime, javax.faces.Number, javax.faces.Boolean, javax.faces.Byte, javax.faces.Character, javax.faces.Double, javax.faces.Float, javax.faces.Integer, javax.faces.Long, javax.faces.Short, javax.faces.BigDecimal et javax.faces.BigInteger.

JSF effectue une conversion implicite des données lorsque celle ci correspond à un type primitif ou à BigDecimal ou BigInteger en utilisant les convertisseurs appropriés.

Deux convertisseurs sont proposés en standard pour mettre en oeuvre des conversions qui ne correspondent pas à des types primitifs :

- le tag convertNumber : utilise le convertisseur javax.faces.Number
- le tag convertDateTime : utilise le convertisseur javax.faces.DateTime

39.11.1. Le tag <convertNumber>

Ce tag permet d'ajouter à un composant un convertisseur de valeur numérique.

Ce tag possède les attributs suivants :

Attributs	Rôle
type	type de valeur. Les valeurs possibles sont number (par défaut), currency et percent
pattern	motif de formatage qui sera utilisé par une instance de java.text.DecimalFormat
maxFractionDigits	nombre maximum de chiffres composant la partie décimale
minFractionDigits	nombre minimum de chiffres composant la partie décimale
maxIntegerDigits	nombre maximum de chiffres composant la partie entière
minIntegerDigits	nombre minimum de chiffres composant la partie entière
integerOnly	booléen qui précise si uniquement la partie entière est prise en compte (false par défaut)
groupingUsed	booléen qui précise si le séparateur de groupe d'unité est utilisé (true par défaut)
locale	objet de type java.util.Locale permettant de définir la locale à utiliser pour les conversions
currencyCode	code de la monnaie utilisée pour la conversion
currencySymbol	symbole de la monnaie utilisé pour la conversion

Exemple :

```
<p>valeur1 = <h:outputText value="#{convert.prix}">
<f:convertNumber type="currency"/>
</h:outputText>
</p>

<p>valeur2 = <h:outputText value="#{convert.poids}">
<f:convertNumber type="number"/>
</h:outputText>
</p>

<p>valeur3 = <h:outputText value="#{convert.ratio}">
<f:convertNumber type="percent"/>
</h:outputText>
</p>

<p>valeur4 = <h:outputText value="#{convert.prix}">
<f:convertNumber integerOnly="true" maxIntegerDigits="2"/>
</h:outputText>
</p>

<p>valeur5 = <h:outputText value="#{convert.prix}">
<f:convertNumber pattern="#.##"/>
</h:outputText>
</p>
```

Le code bean utilisé dans cet exemple est le suivant :

Exemple :

```
package com.jmd.test.jsf;

public class Convert {
  private int poids;
  private float prix;
  private float ratio;

  public Convert() {
    super();
    this.poids = 12345;
    this.prix = 1234.56f ;
    this.ratio = 0.12f ;
  }


  public int getPoids() {
```

```
    return poids;
  }

  public void setPoids(int poids) {
    this.poids = poids;
  }
  public float getRatio() {
    return ratio;
  }

  public void setRatio(float ratio) {
    this.ratio = ratio;
  }

  public float getPrix() {
    return prix;
  }

  public void setPrix(float prix) {
    this.prix = prix;
  }
}
```

valeur1 = 1 234,56 €

valeur2 = 12 345

valeur3 = 12%

valeur4 = 34,56

valeur5 = 1234,56

39.11.2. Le tag <convertDateTime>

Ce tag permet d'ajouter à un composant un convertisseur de valeurs temporelles.

Ce tag possède les attributs suivants :

Attributs	Rôle
Type	type de valeur. Les valeurs possibles sont date (par défaut), time et both
dateStyle	style prédéfini de la date. Les valeurs possibles sont short, medium, long, full ou default
timeStyle	style prédéfini de l'heure. Les valeurs possibles sont short, medium, long, full ou default
Pattern	motif de formatage qui sera utilisé par une instance de java.text.SimpleDateFormat
Locale	objet de type java.util.Locale permettant de définir la locale à utiliser pour les conversions
timeZone	objet de type java.util.TimeZone utilisé lors des conversions

Exemple :

```
<p>Date1 = <h:outputText value="#{convertDate.dateNaiss}">
<f:convertDateTime pattern="MM/yyyy"/>
</h:outputText>
</p>

<p>Date2 = <h:outputText value="#{convertDate.dateNaiss}">
<f:convertDateTime pattern="EEE, dd MMM yyyy"/>
</h:outputText>
```

```
</p>

<p>Date3 = <h:outputText value="#{convertDate.dateNaiss}">
<f:convertDateTime pattern="dd/MM/yyyy"/>
</h:outputText>
</p>

<p>Date4 = <h:outputText value="#{convertDate.dateNaiss}">
<f:convertDateTime dateStyle="full"/>
</h:outputText>
</p>
```

Le code du bean utilisé comme source dans cet exemple est le suivant :

Exemple :

```
package com.jmd.test.jsf;

import java.util.Date;

public class ConvertDate {
  private Date dateNaiss;

  public ConvertDate() {
    super();
    this.dateNaiss = new Date();
  }

  public Date getDateNaiss() {
    return dateNaiss;
  }

  public void setDateNaiss(Date dateNaiss) {
    this.dateNaiss = dateNaiss;
  }
}
```

Résultat :

Date1 = 06/2005

Date2 = mer., 15 juin 2005

Date3 = 15/06/2005

Date4 = mercredi 15 juin 2005

39.11.3. L'affichage des erreurs de conversions

Les messages d'erreurs issus de ces conversions peuvent être affichés en utilisant les tag <message> ou <messages>.

Par défaut, ils contiennent une description : « Conversion error occured ».

Pour modifier ce message par défaut ou l'internationnaliser, il faut définir une clé javax.faces.component.UIInput.CONVERSION dans le fichier properties de définition des chaînes de caractères.

Exemple :

```
javax.faces.component.UIInput.CONVERSION=La valeur saisie n'est pas correctement formatée.
```

39.11.4. L'écriture de convertisseurs personnalisés

JSF fournit en standard des convertisseurs pour les types primitifs et quelques objets de base. Il peut être nécessaire de développer son propre convertisseur pour des besoins spécifiques.

Pour écrire son propre convertisseur, il faut définir une classe qui implémente l'interface Converter. Cette interface définit deux méthodes :

- Object getAsObject(FacesContext context, UIComponent component, String newValue) : cette méthode permet de convertir une chaîne de caractères en objet
- String getAsString(FacesContext context, UIComponent component, Object value) : cette méthode permet de convertir un objet en chaîne de caractères

La méthode getAsObject() doit lever une exception de type ConverterException si une erreur de conversion est détectée dans les traitements.

La suite de ce chapitre sera développée dans une version future de ce document

39.12. La validation des données

JSF propose en standard un mécanisme de validation des données. Celui ci repose sur un ensemble de classes qui permettent de faire des vérifications standards. Il est possible de définir ces propres classes de validation pour répondre à des besoins spécifiques.

La validation peut se faire de deux façons : au niveau de certains composants ou avec des classes spécialement développées pour des besoins spécifiques. Ces classes sont attachables à un composant et sont réutilisables. Ces validations sont effectuées côté serveur.

Les validators sont enregistrés sur des composants. Ce sont des classes qui utilisent des données pour effectuer des opérations de validation de la valeur des données : contrôle de présence, de type de données, de plage de valeurs, de format, ...

39.12.1. Les classes de validation standard

Toutes ces classes implémentent l'interface javax.faces.validator.Validator. JSF propose en standard plusieurs classes pour la validation :

- deux classes de validation sur une plage de données : LongRangeValidator et DoubleRangeValidator
- une classe de validation de la taille d'une chaîne de caractères : LengthValidator

Pour faciliter l'utilisation de ces classes, la bibliothèque de tags personnalisées core propose des tags dédiés à la mise en oeuvre de ces classes :

- validateDoubleRange : utilise la classe DoubleRangeValidator
- validateLongRange : utilise la classe LongRangeValidator
- validateLength : utilise la classe LengthValidator

Ces trois tags possèdent deux attributs nommés minimum et maximum qui permettent de préciser respectivement la valeur de début et de fin selon le Validator utilisé. L'un, l'autre ou les deux attributs peuvent être utilisés.

L'ajout d'une validation sur un contrôle peut se faire de plusieurs manières :

- ajout d'une ou plusieurs validations directement dans la JSP
- ajout par programmation d'une validation en utilisant la méthode addValidator().
- certaines implémentations de composants peuvent contenir des validations implicites.

Pour ajouter une validation à un composant dans la JSP , il suffit d'insérer le tag de validation dans le corps du tag du composant.

Exemple :

```
<h:inputText id="nombre" converter="#{Integer}" required="true"
  value="#{saisieDonnees.nombre}">
  <f:validate_longrange minimum="1" maximum="9" />
</h:inputText>
```

Certaines implémentations de composants peuvent contenir des validations implicites en fonction du contexte. C'est par exemple le cas du composant <inputText> qui, lorsque que son attribut required est à true, effectue un contrôle de présence de données saisies.

Exemple :

```
<h:inputText id=”nombre” converter=”#{Integer}” required=”true”
  value=”#{saisieDonnees.nombre}”/>
```

Toutes les validations sont faites côté serveur dans la version courante de JSF.

Les messages d'erreurs issus de ces conversions peuvent être affichés en utilisant les tags <message> ou <messages>.

Ils contiennent une description par défaut selon le validator utilisé commençant par « Validation error : ».

Pour modifier ce message par défaut ou l'internationnaliser, il faut définir une clé dédiée dans le fichier properties de définition des chaînes de caractères. Les clés définies sont les suivantes :

- javax.faces.component.UIInput.REQUIRED
- javax.faces.validator.NOT_IN_RANGE
- javax.faces.validator.DoubleRangeValidator.MAXIMUM
- javax.faces.validator.DoubleRangeValidator.TYPE
- javax.faces.validator.DoubleRangeValidator.MINIMUM
- javax.faces.validator.LongRangeValidator.MAXIMUM
- javax.faces.validator.LongRangeValidator.MINIMUM
- javax.faces.validator.LongRangeValidator.TYPE
- javax.faces.validator.LengthValidator.MAXIMUM
- javax.faces.validator.LengthValidator.MINIMUM

39.12.2. Contourner la validation

Dans certains cas, il est nécessaire d'empêcher la validation. Par exemple, dans une page de saisie d'informations disposant d'un bouton « Valider » et « Annuler ». La validation doit être opérée lors d'un clic sur le bouton « Valider » mais ne doit pas l'être lors d'un clic sur le bouton « Annuler ».

Pour chaque composant dont l'action doit être exécutée sans validation, il faut mettre l'attribut immediate du composant à true.

Exemple :

```
<h:commandButton value="Annuler" action="annuler" immediate="true"/>
```

39.12.3. L'écriture de classes de validation personnalisées

JSF fournit en standard des classes de validation de base. Il peut être nécessaire de développer ses propres classes de validation pour des besoins spécifiques.

Pour écrire sa propre classe de validation, il faut définir une classe qui implémente l'interface javax.faces.validator.Validator. Cette interface définit une seule méthode :

- public void validate(FacesContext context, UIComponent component, Object toValidate) : cette méthode permet de réaliser les traitements de validation

Elle attend en paramètre :

- un objet de type FacesContext qui permet d'accéder au contexte de l'application jsf
- un objet de type UICOmponent qui contient une référence sur le composant dont la donnée est à valider
- un objet de type Object qui encapsule la valeur de la données à valider.

La méthode validate() doit levée une exception de type ValidatorException si une erreur dans les traitements de validation est détectée.

Exemple :

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

import com.sun.faces.util.MessageFactory;

public class NumeroDeSerieValidator implements Validator {

  public static final String CLE_MESSAGE_VALIDATION_IMPOSSIBLE =
    "message.validation.impossible";

  public void validate(FacesContext contexte, UIComponent composant,
    Object objet) throws ValidatorException {
    String valeur = null;
    boolean estValide = false;

    if ((contexte == null) || (composant == null)) {
      throw new NullPointerException();
    }
    if (!(composant instanceof UIInput)) {
      return;
    }

    valeur = objet.toString();

    Pattern p = Pattern.compile("[0-9][0-9]-[0-9][0-9][0-9]",Pattern.MULTILINE);
    Matcher m = p.matcher(valeur);
    estValide = m.matches();

    if (!estValide) {
      FacesMessage errMsg = MessageFactory.getMessage(contexte,
      CLE_MESSAGE_VALIDATION_IMPOSSIBLE);
      throw new ValidatorException(errMsg);
    }

  }
}
```

Dans l'exemple précédent, la valeur à valider doit respecter une expression régulière de la forme deux chiffres, un tiret et trois chiffres.

Si la validation échoue alors il sera nécessaire d'informer l'utilisateur de la raison de l'échec grâce à un message stocké dans le ressourceBundle de l'application.

Exemple :

```
message.validation.impossible=Le format du numéro de série est erroné
```

La valeur du message dans le ressourceBundle peut être obtenue en utilisant la méthode getMessage() de la classe MessageFactory. Cette méthode attend en paramètres le contexte JSF de l'application et la clé du ressourceBundle à extraire. Elle renvoie un objet de type FacesMessages. Il suffit alors simplement de fournir cet objet à la nouvelle instance de la classe ValidatorException.

Pour pouvoir utiliser une classe de validation, il faut la déclarer dans le fichier de configuration.

Exemple :

```
<validator>
  <validator-id>com.jmd.test.jsf.NumeroDeSerie</validator-id>
  <validator-class>com.jmd.test.jsf.NumeroDeSerieValidator</validator-class>
</validator>
```

Le tag <validator−id> permet de définir un identifiant pour la classe de validation. Le tag <validator−class> permet de préciser la classe pleinement qualifiée.

Pour utiliser la classe de validation dans une page, il faut utiliser le tag <validator> en fournissant à l'attribut validatorId la valeur donnée au tag <validator−id> dans le fichier de configuration :

Exemple :

```
<h:panelGrid columns="2">
  <h:outputText value="Numéro de série : " />
  <h:panelGroup>
    <h:inputText value="#{validation.numeroSerie}" id="numeroSerie" required="true">
      <f:validator validatorId="com.jmd.test.jsf.NumeroDeSerie"/>
    </h:inputText>
    <h:message for="numeroSerie"/>
  </h:panelGroup>
</h:panelGrid>
```

La saisie d'un numéro répondant à l'expression régulière et l'appui sur la touche entrée n'affiche aucun message d'erreur :

La saisie d'un numéro ne répondant pas à l'expression régulière affiche le message d'erreur :

39.12.4. La validation à l'aide de bean

Il est possible de définir une méthode dans un bean qui va offrir les services de validation. Cette méthode doit avoir une signature similaire à celle de la méthode validate() de l'interface Validator.

Exemple :

```
package com.jmd.test.jsf;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;

import com.sun.faces.util.MessageFactory;

public class Validation {
  public static final String CLE_MESSAGE_VALIDATION_IMPOSSIBLE =
    "message.validation.impossible";

  private String numeroSerie;

  public String getNumeroSerie() {
    return numeroSerie;
  }

  public void setNumeroSerie(String numeroSerie) {
    this.numeroSerie = numeroSerie;
  }

  public void valider(FacesContext contexte, UIComponent composant, Object objet) {
    String valeur = null;
    boolean estValide = false;

    if ((contexte == null) || (composant == null)) {
      throw new NullPointerException();
    }
    if (!(composant instanceof UIInput)) {
      return;
    }

    valeur = objet.toString();

    Pattern p = Pattern.compile("[0-9][0-9]-[0-9][0-9][0-9]",Pattern.MULTILINE);
    Matcher m = p.matcher(valeur);
    estValide = m.matches();

    if (!estValide) {
      FacesMessage errMsg = MessageFactory.getMessage(contexte,
      CLE_MESSAGE_VALIDATION_IMPOSSIBLE);
      throw new ValidatorException(errMsg);
    }
```

```
  }
}
```

Pour utiliser cette méthode, il faut utiliser l'attribut validator et lui fournir en paramètre une expression qui désigne la méthode d'une instance du bean

Exemple :

```
<h:panelGrid columns="2">
  <h:outputText value="Numéro de série : " />
  <h:panelGroup>
    <h:inputText value="#{validation.numeroSerie}" id="numeroSerie"
      required="true" validator="#{validation.valider}" />
    <h:message for="numeroSerie"/>
  </h:panelGroup>
</h:panelGrid>
```

Cette approche est particulièrement utile pour des besoins spécifiques à une application car sa mise en oeuvre est difficilement portable d'une application à une autre.

39.12.5. Validation entre plusieurs composants

De base, le modèle de validation des données proposé par JSF repose sur une validation unitaire de chaque composant. Il est cependant fréquent d'avoir besoin de faire une validation en fonction des données d'un ou plusieurs autres composants.

Pour réaliser ce genre de tâche, il faut définir un backing bean qui aura accès à chacun des composants nécessaire aux traitements et de définir dans ce bean une méthode qui va réaliser les traitements de validation.

Exemple :

```
package com.jmd.test.jsf;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;

import com.sun.faces.util.MessageFactory;

public class Validation {
  public static final String CLE_MESSAGE_VALIDATION_IMPOSSIBLE =
    "message.validation.impossible";

  private String numeroSerie;
  private String cle;
  private UIInput cleInput;
  private UIInput numeroSerieInput;

  public String getNumeroSerie() {
    return numeroSerie;
  }

  public void setNumeroSerie(String numeroSerie) {
    this.numeroSerie = numeroSerie;
  }
  public void valider(FacesContext contexte, UIComponent composant, Object objet) {
    String valeur = null;
    boolean estValide = false;
```

```
    if ((contexte == null) || (composant == null)) {
      throw new NullPointerException();
    }
    if (!(composant instanceof UIInput)) {
      return;
    }

    valeur = objet.toString();

    Pattern p = Pattern.compile("[0-9][0-9]-[0-9][0-9][0-9]",Pattern.MULTILINE);
    Matcher m = p.matcher(valeur);
    estValide = m.matches();

    if (!estValide) {
      FacesMessage errMsg = MessageFactory.getMessage(contexte,
      CLE_MESSAGE_VALIDATION_IMPOSSIBLE);
      throw new ValidatorException(errMsg);
    }
  }

  public void validerCle(FacesContext contexte, UIComponent composant, Object objet) {
    System.out.println("validerCle");

    String valeurNumero = numeroSerieInput.getLocalValue().toString();
    String valeurCle = cleInput.getLocalValue().toString();
    boolean estValide = false;
    if (contexte == null) {
      throw new NullPointerException();
    }

    Pattern p = Pattern.compile("[0-9][0-9]-[0-9][0-9][0-9]",Pattern.MULTILINE);
    Matcher m = p.matcher(valeurNumero);
    estValide = m.matches() && valeurCle.equals("789");

    System.out.println("estValide="+estValide);
    if (!estValide) {
      FacesMessage errMsg = MessageFactory.getMessage(contexte,
      CLE_MESSAGE_VALIDATION_IMPOSSIBLE);
      throw new ValidatorException(errMsg);
    }
  }

  public String getCle() {
    return cle;
  }

  public void setCle(String cle) {
    this.cle = cle;
  }

  public UIInput getCleInput() {
    return cleInput;
  }

  public void setCleInput(UIInput cleInput) {
    this.cleInput = cleInput;
  }

  public UIInput getNumeroSerieInput() {
    return numeroSerieInput;
  }

  public void setNumeroSerieInput(UIInput numeroSerieInput) {
    this.numeroSerieInput = numeroSerieInput;
  }
}
```

Il suffit alors d'ajouter un champ caché dans la vue sur lequel la classe de validation sera appliquée.

Exemple :

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ page language="java" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<f:view>
<head>
  <title>Tests de validation</title>
</head>
<body bgcolor="#FFFFFF">
  <h:form>
  <h2>Tests de validation</h2>

  <h:panelGrid columns="2">
    <h:outputText value="Numéro de série : " />
    <h:panelGroup>
      <h:inputText value="#{validation.numeroSerie}" id="numeroSerie"
        required="true" binding="#{validation.numeroSerieInput}" />
      <h:message for="numeroSerie"/>
    </h:panelGroup>
    <h:outputText value="clé : " />
    <h:panelGroup>
      <h:inputText value="#{validation.cle}" id="cle" binding="#{validation.cleInput}"
        required="true" />
      <h:message for="validationCle"/>
    </h:panelGroup>
  </h:panelGrid>

  <h:inputHidden id="validationCle" validator="#{validation.validerCle}" value="nul"/>

  <h:commandButton value="Valider" action="submit"/>

  </h:form>
</body>
</f:view>
</html>
```

39.12.6. Ecriture de tags pour un convertisseur ou un valideur de données

L'écriture de tag personnalisé facilite l'utilisation d'un convertisseur ou d'un valideur et permet de leur fournir des paramètres.

Il faut définir une classe nommée handler qui va contenir les traitements du tag. Cette classe doit hériter d'une sous classe dédiée selon le type d'élément que va représenter le tag :

- ConverterTag : si le tag concerne un convertisseur
- ValidatorTag : si le tag concerne un validateur
- UIComponentTag et UIComponentBodyTag : si le tag concerne un composant

Le handler est un bean dont une propriété doit correspondre à chaque attribut défini dans le tag.

Pour pouvoir utiliser un tag personnalisé, il faut définir un fichier .tld

Ce fichier au format XML défini dans les spécifications des JSP permet de fournir des informations sur la bibliothèque de tags personnalisés notamment la version des spécifications utilisées et des informations sur chaque tag.

Enfin, il est nécessaire de déclarer l'utilisation de la bibliothèque de tags personnalisés dans la JSP.

39.12.6.1. Ecriture d'un tag personnalisé pour un convertisseur

Il faut définir un handler pour le tag qui est un bean qui hérite de la classe ConverterTag.

Il est important dans le constructeur du handler de faire un appel à la méthode setConverterId() en lui passant un id défini dans le fichier de configuration de l'application JSF.

Il faut redéfinir la méthode release() dont les traitements vont permettre de réinitialiser les propriétés de la classe. Ceci est important car l'implémentation utilisée pour utiliser un pool pour ces objets afin d'augmenter les performances. La méthode release() est dans ce cas utilisée pour recycler les instances du pool non utilisées.

Il faut ensuite redéfinir la méthode createConverter() qui va permettre la création d'une instance du converter en utilisant les éventuels valeurs des attributs du tag.

La valeur fournie à un attribut d'un tag pour être soit un littéral soit une expression dont le contenu devra être évalué pour connaître la valeur à un instant donné.

La suite de ce chapitre sera développée dans une version future de ce document

39.12.6.2. Ecriture d'un tag personnalisé pour un valideur

L'écriture d'un tag personnalisé pour un valideur suit les mêmes règles que pour un convertisseur. La grande différence est que la classe handler doit hériter de la classe ValidatorTag. La méthode à appeler dans le constructeur est la méthode setValidatorId() et la méthode à redéfinir pour créer une instance du valideur est la méthode createValidator().

La suite de ce chapitre sera développée dans une version future de ce document

39.13. Sauvegarde et restauration de l'état

JSF sauvegarde l'état de chaque élément présent dans la vue : les composants, les convertisseurs, les valideurs, ... pourvu que ceux ci mettent en oeuvre un mécanisme adéquat.

Ces états sont stockés dans un champ de type hidden dans la vue pour permettre l'échange de ces informations entre deux

requêtes si l'application est configurée dans ce sens dans le fichier de configuration.

Ce mécanisme peut prendre deux formes :

- la classe qui encapsule l'élément peut implémenter l'interface Serializable
- la classe qui encapsule l'élément peut implémenter l'interface StateHolder

Dans le premier cas, c'est le mécanisme standard de la sérialisation qui sera utilisé. Il nécessite donc très peux voir aucun code particulier si les champs de la classe sont tous d'un type qui est sérialisable.

L'implémentation de l'interface StateHolder nécessite la définition des deux méthodes définies dans l'interface (saveState() et restoreState()) et la présence d'un constructeur par défaut. Cette approche peut être intéressante pour obtenir un contrôle très fin de la sauvegarde et de la restauration de l'état.

La méthode saveState(FacesContext) renvoie un objet serialisable qui va contenir les données de l'état à sauvegarder. La méthode restoreState(FacesContext, Object) effectue l'opération inverse.

Il est aussi nécessaire de définir une propriété nommée transient de type booléen qui précise si l'état doit être sauvegardé ou non.

Si l'élément n'implémente pas l'interface Serializable ou StateHolder alors son état n'est pas sauvegardé entre deux échanges de la vue.

39.14. Le système de navigation

Une application de type web se compose d'un ensemble de pages dans lesquelles l'utilisateur navigue en fonction de ces actions.

Un système de navigation standard peut être facilement mis en oeuvre avec JSF grâce à un paramétrage au format XML dans le fichier de configuration de l'application.

Le système de navigation assure la gestion de l'enchainement des pages en utilisant des actions. Les règles de navigation sont des chaînes de caractères qui sont associés à une page d'origine et qui permet de déterminer la page de résultat. Toutes ces règles sont contenues dans le fichier de configuration face−config.xml.

La déclaration de ce système de navigation ressemble à celle utilisée dans le framework Struts.

Le système de navigation peut être statique ou dynamique. Dans ce dernier cas, des traitements particuliers doivent être mis en place pour déterminer la cible de la navigation.

Exemple :

```
...
  <navigation−rule>
    <from−view−id>/login.jsp</from−view−id>
    <navigation−case>
      <from−outcome>login</from−outcome>
      <to−view−id>/accueil.jsp</to−view−id>
    </navigation−case>
  </navigation−rule>
...
```

La tag <navigation−rule> permet de préciser des règles de navigation.

La tag <from−view−id> permet de préciser qu'elle est la page concernée. Ce tag n'est pas obligatoire : sans sa présence, il est possible définir une règle de navigation applicable à toutes les pages JSF de l'application.

Exemple :

```
<navigation−rule>
```

```
  <navigation-case>
    <from-outcome>logout</from-outcome>
    <to-view-id>/logout.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Il est aussi possible de désigner un ensemble de page dans le tag <from−view−id> en utilisant le caractère * dans la valeur du tag. Ce caractère * ne peut être utilisé qu'une seule fois dans la valeur du tag et il doit être en dernière position.

Exemple :

```
<from-view-id>/admin/*</from-view-id>
```

Le tag <navigation−case> permet de définir les différents cas.

La valeur du tag <from−outcome> doit correspondre au nom d'une action.

Le tag <to−view−id> permet de préciser la page qui sera affichée. L'url fournie comme valeur doit commencer par un slash et doit préciser une page possédant une extension brute (ne surtout pas mettre une url utilisée par la servlet faisant office de contrôleur).

Le tag <redirect/> inséré juste après le tag <to−view−id> permet de demander la redirection vers la page au navigateur de l'utilisateur.

La gestion de la navigation est assurée par une instance de la classe NavigationHandler, gérée au niveau de l'application. Ce gestionnaire utilise la valeur d'un attribut action d'un composant pour déterminer la page suivante et faire la re−direction vers la page adéquat en fonction des informations fournies dans le fichier de configuration.

La valeur de l'attribut action peut être statique : dans ce cas la valeur est en dur dans le code de la vue

Exemple :

```
<h:commandButton action="login"/>
```

La valeur de l'attribut action peut être dynamique : dans ce cas la valeur est déterminée par l'appel d'une méthode d'un bean

Exemple :

```
<h:commandButton action="#{login.verifierMotDePasse}"/>
```

Dans ce cas, la méthode appelée ne doit pas avoir de paramètres et doit retourner une chaîne de caractères définie dans la navigation du fichier de configuration.

Lors des traitements par le NavigationHandler, si aucune action ne trouve de correspondance dans le fichier de configuration pour la page alors la page est simplement réaffichée.

39.15. La gestion des événements

Le modèle de gestion de événements de JSF est similaire est celui utilisé dans les JavaBeans : il repose sur les Listener et les Event pour traiter les événements générés dans les composants graphiques suite aux actions de l'utilisateur.

Un objet de type Event encapsule le composant à l'origine de l'événement et des données relatives à cet événement.

Pour être notifié d'un événement particulier, il est nécessaire d'enregistrer un objet qui implément l'interface Listener auprès du composant conserné.

Lors de certaines actions de l'utilisateur, un événement est émis.

L'implementation JSF propose deux types d'événements :

- Value changed : ces événements sont émis lors du changement de la valeur d'un composant de type UIInput, UISelectOne, UISelectMany, et UISelectBoolean
- Action : ces événements sont émis lors d'un clic sur un hyperlien ou un bouton qui sont des composants de type UICommand

JSF propose de transposer le modèle de gestion des événements des interfaces graphiques des applications standalone aux applications de type web utilisant JSF.

La gestion des événements repose donc sur deux types d'objets

- Event : classe qui encapsule l'événement lui même
- Listener : classe qui va encapsuler les traitements à réaliser pour un type d'événements

Comme pour les interfaces graphiques des applications standalone, la classe de type Listener doit s'enregistrer auprès du composant concerné. Lorsque celui ci émet un événement suite à une action de l'utilisateur, il appelle le Listener enregistré en lui fournissant en paramètre un objet de type Event.

Exemple :

```
<h:selectOneMenu ... valueChangeListener="#{choixLangue.langueChangement}">
  ...
</h:selectOneMenu>
```

JSF supporte trois types d'événements :

- les changements de valeurs : concernent les composants qui permettent la saisie ou la selection d'une valeur et que cette valeur change
- les actions : concernent un clic sur un bouton (commandButton) ou un lien (commandLink)
- les événements liés au cycle de vie : ils sont émis par le framework JSF durant le cyle de vie des traitements

Les traitements des listeners peuvent affecter la suite des traitements du cycle de vie de plusieurs manières :

- par défaut, laisser ces traitements se poursuivre
- demander l'exécution immédiate de la dernière étape en utilisant la méthode FacesContext.renderResponse()
- arrêter les traitements du cyle de vie en utilisant la méthode FacesContext.responseComplete()

39.15.1. Les événements liés à des changements de valeur

Il y a deux façons de préciser un listener de type valueChangeListener sur un composant :

- utiliser l'attribut valueChangeListener
- utiliser le tag valueChangeListener

L'attribut valueChangeListener permet de préciser une expression qui désigne une méthode qui sera exécutée durant les traitements du cyle de vie de la requête. Pour que ces traitements puissent être déclenchés, il faut soumettre la page.

Exemple :

```
<h:selectOneMenu value="#{choixLangue.langue}" onchange="submit()"
  valueChangeListener="#{choixLangue.langueChangement}">
  <f:selectItems value="#{choixLangue.langues}"/>
</h:selectOneMenu>
```

La méthode ne renvoie aucune valeur et attend en paramètre un objet de type ValueChangeEvent.

Exemple :

```
package com.jmd.test.jsf;

import java.util.Locale;
import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;

public class ChoixLangue {
  private static final String LANGUE_FR = "Français";
  private static final String LANGUE_EN = "Anglais";
  private String langue = LANGUE_FR;

  private SelectItem[] langueItems = {
    new SelectItem(LANGUE_FR, "Français"),
    new SelectItem(LANGUE_EN, "Anglais") };

  public SelectItem[] getLangues() {
    return langueItems;
  }

  public String getLangue() {
    return langue;
  }

  public void setLangue(String langue) {
    this.langue = langue;
  }

  public void langueChangement(ValueChangeEvent event) {
    FacesContext context = FacesContext.getCurrentInstance();
    System.out.println("Changement de la langue : "+event.getNewValue());
    if (LANGUE_FR.equals((String) event.getNewValue()))
      context.getViewRoot().setLocale(Locale.FRENCH);
    else
      context.getViewRoot().setLocale(Locale.ENGLISH);
    }
  }
}
```

La classe ValueChangeEvent possède plusieurs méthodes utiles :

Méthode	Rôle
UIComponent getComponent()	renvoie le composant qui a généré l'événement
Object getNewValue()	renvoie la nouvelle valeur (convertie et validée)
Object getOldValue()	renvoie la valeur précédente

Le tag valueChangeListener permet aussi de préciser un listener. Son attribut type permet de préciser une classe implémentant l'interface ValueChangeListener.

Exemple :

```
<h:selectOneMenu value="#{choixLangue.langue}" onchange="submit()">
  <f:valueChangeListener type="com.jmd.test.jsf.ChoixLangueListener"/>
  <f:selectItems value="#{choixLangue.langues}"/>
</h:selectOneMenu>
```

Une telle classe doit définir une méthode processValueChange() qui va contenir les traitements exécutés en réponse à l'événement.

Exemple :

```
package com.jmd.test.jsf;

import java.util.Locale;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;

public class ChoixLangueListener implements ValueChangeListener {

  private static final String LANGUE_FR = "Français";

  private static final String LANGUE_EN = "Anglais";

  public void processValueChange(ValueChangeEvent event)
    throws AbortProcessingException {
    FacesContext context = FacesContext.getCurrentInstance();
    System.out.println("Changement de la langue : " + event.getNewValue());
    if (LANGUE_FR.equals((String) event.getNewValue()))
      context.getViewRoot().setLocale(Locale.FRENCH);
    else
      context.getViewRoot().setLocale(Locale.ENGLISH);
    }
  }
}
```

39.15.2. Les événements liés à des actions

Les actions sont des clics sur des boutons ou des liens. Le clic sur un composant de type commandLink ou commandButton déclenche automatiquement la soumission de la page.

Il y a deux façons de préciser un listener de type actionListener sur un composant :

- utiliser l'attribut actionListener
- utiliser le tag actionListener

L'attribut actionListener permet de préciser une expression qui désigne une méthode qui sera exécutée durant les traitements du cyle de vie de la requête.

Exemple :

```
<table align="center" width="50%">
  <tr>
    <td width="50%"><h:commandButton image="images/bouton_valider.gif"
      actionListener="#{saisieDonnees.traiterAction}"
      id="Valider" />
    </td>
    <td><h:commandButton image="images/bouton_annuler.gif"
      actionListener="#{saisieDonnees.traiterAction}"
      id="Annuler"/>
    </td>
  </tr>
</table>
```

Cette méthode attend en paramètre un objet de type ActionEvent.

Exemple :

```
package com.jmd.test.jsf;

import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
```

```
public class SaisieDonnees {

  public void traiterAction(ActionEvent e) {

    FacesContext context = FacesContext.getCurrentInstance();

    String clientId = e.getComponent().getClientId(context);
    System.out.println("traiterAction : clientId=" + clientId);

  }
}
```

Le tag valueChangeListener permet aussi de préciser un listener. Son attribut type permet de préciser une classe implémentant l'interface ValueChangeListener.

Exemple :

```
<table align="center" width="50%">
  <tr>
    <td width="50%"><h:commandButton image="images/bouton_valider.gif" id="Valider" >
        <f:actionListener type="com.jmd.test.jsf.SaisieDonneesListener"/>
      </h:commandButton>
      </td>
    <td><h:commandButton image="images/bouton_annuler.gif" id="Annuler">
        <f:actionListener type="com.jmd.test.jsf.SaisieDonneesListener"/>
      </h:commandButton>
    </td>
  </tr>
</table>
```

Une telle classe doit définir la méthode processAction() définie dans l'interface.

Exemple :

```
package com.jmd.test.jsf;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;

public class SaisieDonneesListener implements ActionListener {

  public void processAction(ActionEvent e) throws AbortProcessingException {
    FacesContext context = FacesContext.getCurrentInstance();

    String clientId = e.getComponent().getClientId(context);
    System.out.println("processAction : clientId=" + clientId);

  }
}
```

39.15.3. L'attribut immediate

L'attribut immediate permet de demander les traitements immédiat des listeners.

Par exemple, sur une page un composant possède un attribut required et un second possède un listener. Les traitements du second doivent pouvoir être réalisés sans que le premier composant n'affiche un messsage d'erreur lié à sa validation.

Le cycle de traitement de la requête est modifié lorsque l'attribut immediate est positionné dans un composant. Dans ce cas, les données du composant sont converties et validées si nécessaire puis les traitements du listener sont executés à la place de l'étape « Process validations » (juste après l'étape Apply Request Value).

Exemple :

```
<h:selectOneMenu value="#{choixLangue.langue}" onchange="submit()" immediate="true">
  <f:valueChangeListener type="com.jmd.test.jsf.ChoixLangueListener"/>
  <f:selectItems value="#{choixLangue.langues}"/>
</h:selectOneMenu>
```

Par défaut, ceci modifie l'ordre d'execution des traitements du cycle de vie mais n'empêche pour les traitements prévus de s'exécuter. Pour les inhiber, il est necessaire de demande au framework JSF d'interrompre les traitements du cycle de vie en utilisant la méthode renderResponse() du context.

Exemple :

```
 public void langueChangement(ValueChangeEvent event) {
    FacesContext context = FacesContext.getCurrentInstance();
    System.out.println("Changement de la langue : " + event.getNewValue());
    if (LANGUE_FR.equals((String) event.getNewValue())) {
      context.getViewRoot().setLocale(Locale.FRENCH);
    } else {
      context.getViewRoot().setLocale(Locale.ENGLISH);
    }
    context.renderResponse();
  }
```

Le mode de fonctionnement est le même avec les actionListener hormis le fait que l'appel à la méthode renderResponse() est inutile puisqu'il est automatiquement fait par le framework.

39.15.4. Les événements liés au cycle de vie

Le framework émet des évenements avant et après chaque étape du cycle de vie des requêtes. Ils sont traités par des phaseListeners.

L'enregistrement d'un phaseListener se fait dans le fichier de configuration dans un tag fils <phase−listener> fils du tag <lifecycle> qui doit contenir le nom pleinement qualifié d'une classe.

Exemple :

```
<faces−config>
...

  <lifecycle>
    <phase−listener>com.jmd.test.jsf.PhasesEcouteur</phase−listener>
  </lifecycle>

</faces−config>
```

La classe précisée doit implémenter l'interface javax.faces.event.PhaseListener qui définit trois méthodes :

- getPhaseId() : renvoie un objet de type PhaseId qui permet de préciser à quelle phase se listener correspond
- beforePhase() : traitements à éxécuter avant l'éxécution de la phase
- afterPhase() : traitements à exécuter après l'exécution de la phase

La classe PhaseId définit des constantes permettant d'identifier chacune des phases : PhaseId.RESTORE_VIEW, PhaseId.APPLY_REQUEST_VALUES, PhaseId.PROCESS_VALIDATIONS, PhaseId.UPDATE_MODEL_VALUES, PhaseId.INVOKE_APPLICATION et PhaseId.RENDER_RESPONSE

Elle définit aussi la constante PhaseId.ANY_PHASE qui permet de demander l'application du listener à toutes les phases. Cela peut être très utile lors du débogage.

Exemple :

```
package com.jmd.test.jsf;

import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;

public class PhasesEcouteur implements PhaseListener {

  public void afterPhase(PhaseEvent pe) {
    System.out.println("Apres " + pe.getPhaseId());
  }

  public void beforePhase(PhaseEvent pe) {
    System.out.println("Avant " + pe.getPhaseId());
  }

  public PhaseId getPhaseId() {
    return PhaseId.ANY_PHASE;
  }
}
```

Lors de l'appel de la première page de l'application, les informations suivantes sont affichées dans la sortie standard

Avant RESTORE_VIEW 1
Apres RESTORE_VIEW 1
Avant RENDER_RESPONSE 6
Apres RENDER_RESPONSE 6

Lors d'une soumission de cette page avec une erreur de validation des données, les informations suivantes sont affichées dans la sortie standard

Avant RESTORE_VIEW 1
Apres RESTORE_VIEW 1
Avant APPLY_REQUEST_VALUES 2
Apres APPLY_REQUEST_VALUES 2
Avant PROCESS_VALIDATIONS 3
Apres PROCESS_VALIDATIONS 3
Avant RENDER_RESPONSE 6
Apres RENDER_RESPONSE 6

Lors d'une soumission de cette page sans erreur de validation des données, les informations suivantes sont affichées dans la sortie standard

Avant RESTORE_VIEW 1
Apres RESTORE_VIEW 1
Avant APPLY_REQUEST_VALUES 2
Apres APPLY_REQUEST_VALUES 2
Avant PROCESS_VALIDATIONS 3
Apres PROCESS_VALIDATIONS 3
Avant UPDATE_MODEL_VALUES 4
Apres UPDATE_MODEL_VALUES 4
Avant INVOKE_APPLICATION 5
Apres INVOKE_APPLICATION 5
Avant RENDER_RESPONSE 6
Apres RENDER_RESPONSE 6

39.16. Déploiement d'une application

Une application utilisant JSF s'exécute dans un serveur d'application contenant un conteneur web implémentant les spécifications servlet 1.3 et JSP 1.2 minimum. Une telle application doit être packagée dans un fichier .war.

La compilation des différentes classes de l'application nécessite l'ajout dans le classpath de la bibliothèque servlet.

Elle nécessite aussi l'ajout dans le classpath de la bibliothèque jsf−api.jar de la ou des bibliothèques requises par l'implémentation JSF utilisées.

Ces bibliothèques doivent aussi être disponibles pour le conteneur web qui va éxecuter l'application. Le plus simple est de mettre ces fichiers dans le répertoire WEB−INF/lib

39.17. Un exemple d'application simple

Cette section va développer une petite application constituée de deux pages. La première va demander le nom de l'utilisateur et la seconde afficher un message de bienvenue.

Il faut créer un répertoire, par exemple nommé Test_JSF et créer à l'intérieur la structure de l'application qui correspond à la structure de toute application Web selon les spécifications J2EE, notamment le répertoire WEB−INF avec ces sous répertoires lib et classes.

Il faut ensuite copier les fichiers nécessaire à une utilisation de JSF dans l'application web.

Il suffit de copier*.jar du répertoire lib de l'implémentation de référence vers le répertoire WEB−INF/lib du projet.

Il faut créer un fichier à la racine du projet et le nommer index.htm

Exemple :

```
<html>
  <head>
    <meta http−equiv="Refresh" content= "0; URL=login.faces"/>
    <title>Demarrage de l'application</title>
  </head>
  <body>
    <p>D&eacute;marrage de l'application ...</p>
  </body>
</html>
```

Il faut créer un fichier à la racine du projet et le nommer login.jsp

Exemple :

```
<html>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<head>
  <title>Application de tests avec JSF</title>
</head>
<body>
  <h:form>
    <h3>Identification</h3>
    <table>
      <tr>
        <td>Nom : </td>
        <td><h:inputText value="#{login.nom}"/></td>
      </tr>
      <tr>
        <td>Mot de passe :</td>
        <td><h:inputSecret value="#{login.mdp}"/></td>
      </tr>
      <tr>
        <td colspan="2"><h:commandButton value="Login" action="login"/></td>
      </tr>
    </table>
  </h:form>
</body>
```

```
</f:view>
</html>
```

Il faut créer une nouvelle classe nommée com.jmd.test.jsf.LoginBean et la compiler dans le répertoire WEB−INF/classes.

Exemple :

```
package com.jmd.test.jsf;

public class LoginBean {

  private String nom;
  private String mdp;

  public String getMdp() {
    return mdp;
  }

  public String getNom() {
    return nom;
  }

  public void setMdp(String string) {
    mdp = string;
  }

  public void setNom(String string) {
    nom = string;
  }

}
```

Il faut créer un fichier à la racine du projet et le nommer accueil.jsp : cette page contiendra la page d'accueil de l'application.

Il faut créer un fichier dans le répertoire /WEB−INF et le nommer faces−config.xml

Exemple :

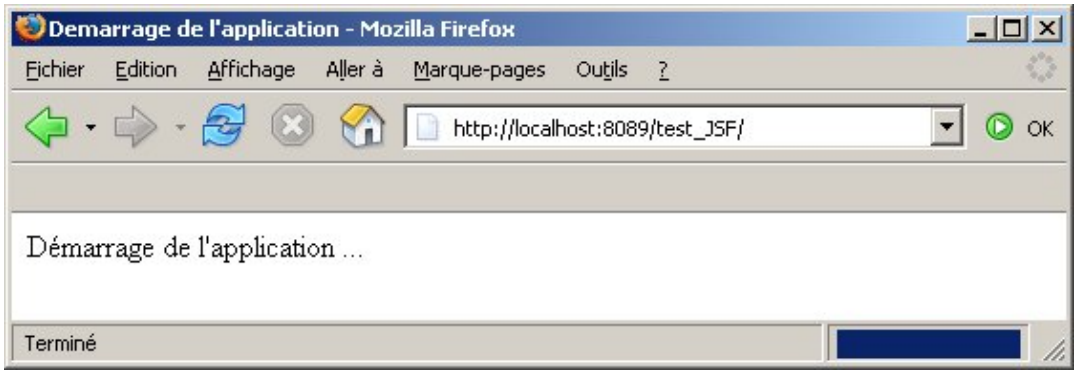
```
<?xml version="1.0"?>

<!DOCTYPE faces−config PUBLIC
"−//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web−facesconfig_1_0.dtd">

<faces−config>
  <navigation−rule>
    <from−view−id>/login.jsp</from−view−id>
    <navigation−case>
      <from−outcome>login</from−outcome>
      <to−view−id>/accueil.jsp</to−view−id>
    </navigation−case>
  </navigation−rule>

  <managed−bean>
    <managed−bean−name>login</managed−bean−name>
    <managed−bean−class>com.jmd.test.jsf.LoginBean</managed−bean−class>
    <managed−bean−scope>session</managed−bean−scope>
  </managed−bean>

</faces−config>
```

Il faut créer un fichier dans le répertoire /WEB−INF et le nommer web.xml

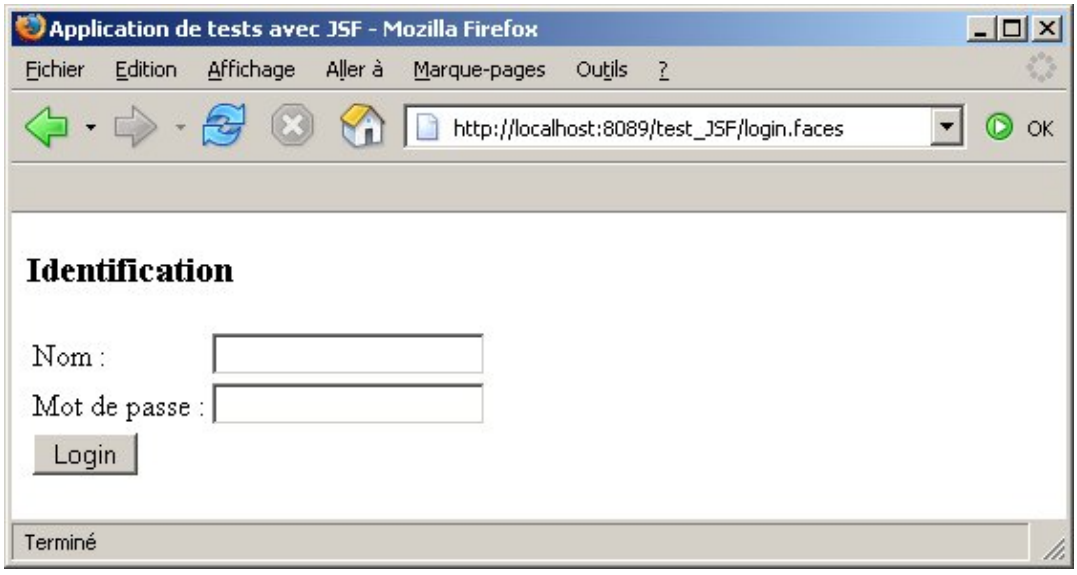
Exemple :

```
<?xml version="1.0"?>

<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.htm</welcome-file>
  </welcome-file-list>

</web-app>
```

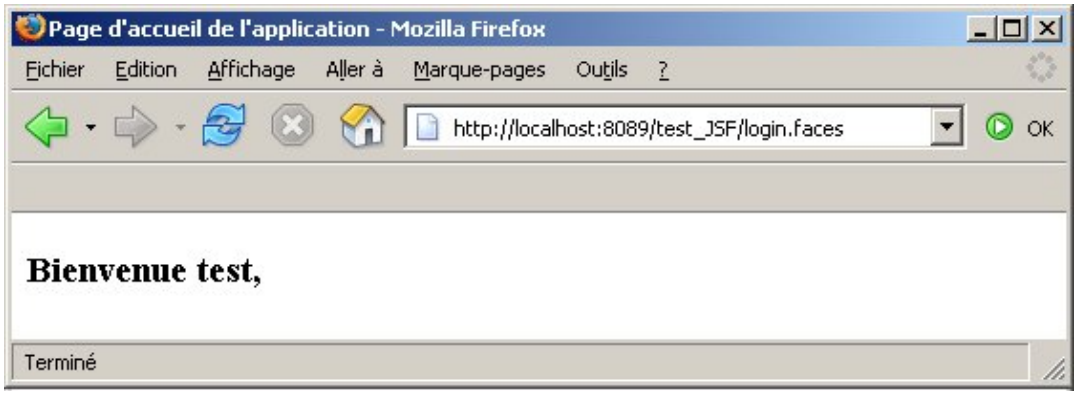
Il suffit alors de démarrer Tomcat, puis d'ouvrir un navigateur et taper l'url http://localhost:8089/test_JSF/ (en remplaçant le port 8089 par celui défini dans Tomcat).



Une fois l'application démarrée, la page de login s'affiche



Il faut saisir un nom par exemple test et cliquer sur le bouton « Login ».



Cette exemple ne met en aucune façon en valeur la puissance de JSF mais permet simplement de mettre en place les éléments minimum pour une application utilisant JSF.

39.18. L'internationnalisation

JSF propose des fonctionnalités qui facilitent l'internationalisation d'une application.

Il faut définir un fichier au format properties qui va contenir la définition des chaînes de caractères. Un tel fichier possède les caractéristiques suivantes :

- le fichier doit avoir l'extension .properties
- il doit être dans le classpath de l'application
- il est composé d'une paire clé=valeur par ligne. La clé permet d'identifier de façon unique la chaîne de caractères

Exemple : le fichier msg.properties

```
login_titre=Application de tests avec JSF
login_identification=Identification
login_nom=Nom
login_mdp=Mot de passe
login_Login=Valider
```

Ce fichier correspond à la langue par défaut. Il est possible de définir d'autre fichier pour d'autres langues. Ces fichiers doivent être avoir le même nom suivi d'un underscore et du code langue défini par le standard ISO 639 avec toujours l'extension .properties.

Exemple :

```
msg.properties
msg_en.properties
msg_de.properties
```

Il faut bien sûre remplacer les valeurs de chaque chaîne par leur traduction correspondante.

Exemple : le fichier msg_en.properties

```
login_titre=Tests of JSF
login_identification=Login
login_nom=Name
login_mdp=Password
login_Login=Login
```

Les langues disponibles doivent être précisées dans le fichier de configuration.

Exemple :

```
<faces-config>
...
  <application>
    <locale-config>
      <default-locale>fr</default-locale>
      <supported-locale>en</supported-locale>
    </locale-config>
  </application>
...
</faces-config>
```

Pour utiliser l'internationnalisation dans les vues, il faut utiliser le tag <f:loadBundle> pour charger le fichier .properties nécessaire. Deux attributs de ce tags sont requis :

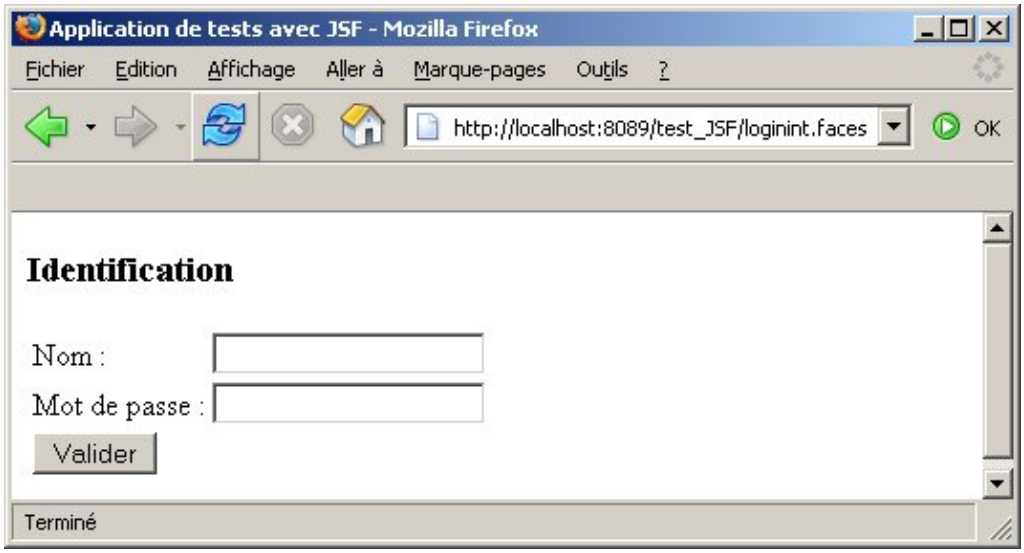
- basename : précise la localisation et le nom de base des fichiers .properties. La notation de la localisation est similaire à celle utilisée pour les packages
- var : précise le nom de la variable qui va contenir les chaînes de caractères

Il ne reste plus qu'à utiliser la variable définie en utilisant la notation avec un point pour la clé de la chaîne dont on souhaite utiliser la valeur.

Exemple :

```
<html>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<f:loadBundle basename="com.jmd.test.jsf.msg" var="msg"/>

<head>
  <title><h:outputText value="#{msg.login_titre}"/></title>
</head>
<body>
  <h:form>
    <h3><h:outputText value="#{msg.login_identification}"/></h3>
    <table>
      <tr>
        <td><h:outputText value="#{msg.login_nom}"/> : </td>
        <td><h:inputText value="#{login.nom}"/></td>
      </tr>
      <tr>
        <td><h:outputText value="#{msg.login_mdp}"/> :</td>
        <td><h:inputSecret value="#{login.mdp}"/></td>
      </tr>
      <tr>
        <td colspan="2"><h:commandButton value="#{msg.login_Login}" action="login"/></td>
      </tr>
    </table>
  </h:form>
</body>
</f:view>
</html>
```

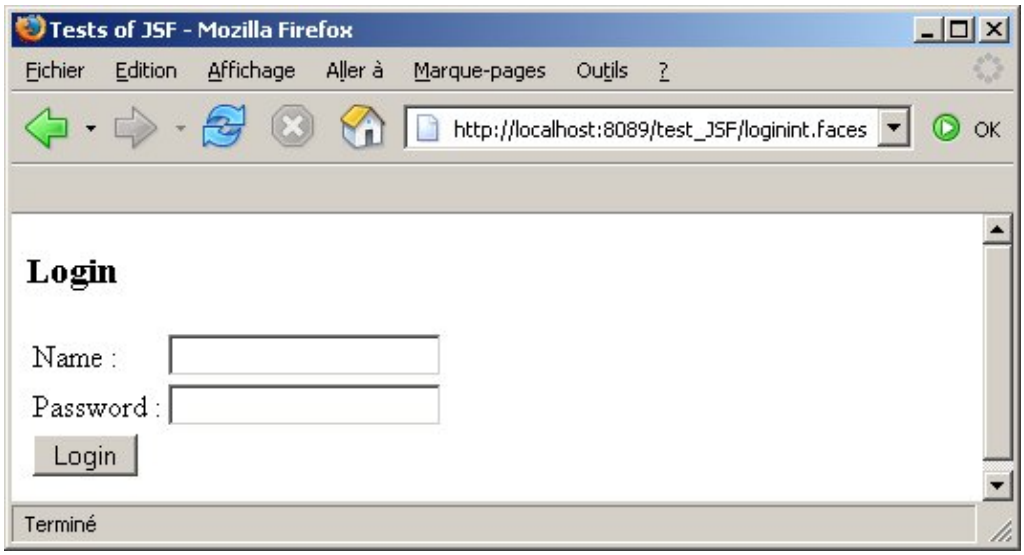
La langue à utiliser est déterminée automatiquement par JSF en fonction des informations contenues dans la propriété Accept-Language de l'en-tête de la requête et du fichier de configuration.



La langue peut aussi être forcée dans l'objet de type view en précisant le code langue dans l'attribut locale.

Exemple :

```
...
         <f:view locale="en">
...
```



Elle peut aussi être déterminée dans le code des traitements. L'exemple suivant va permettre à l'utilisateur de sélectionner la langue utilisée entre français et anglais grace à deux petites icônes cliquables.

Exemple :

```
<html>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<f:loadBundle basename="com.jmd.test.jsf.Messages" var="msg"/>
<head>
<title>Application de tests avec JSF</title>
</head>
<body>
  <h:form>

  <table>
    <tr>
      <td>
        <h:commandLink action="#{langueApp.activerFR}" immediate="true">
```

```
          <h:graphicImage value="images/francais.jpg" style="border: 0px"/>
        </h:commandLink>
      </td>
      <td>
        <h:commandLink action="#{langueApp.activerEN}" immediate="true">
          <h:graphicImage value="images/anglais.jpg" style="border: 0px"/>
        </h:commandLink>
      </td>
      <td width="100%"> </td>
    </tr>
  </table>

  <h3><h:outputText value="#{msg.login_titre}" /></h3>
  <p> </p>
  <h:panelGrid columns="2">
    <h:outputText value="#{msg.login_nom}" />
    <h:panelGroup>
      <h:inputText value="#{login.nom}" id="nom" required="true"
        binding="#{login.inputTextNom}"/>
      <h:message for="nom"/>
    </h:panelGroup>
    <h:outputText value="#{msg.login_mdp}" />
    <h:inputSecret value="#{login.mdp}"/>
    <h:commandButton value="#{msg.login_valider}" action="login"/>
  </h:panelGrid>

  </h:form>
  </body>
</f:view>
</html>
```

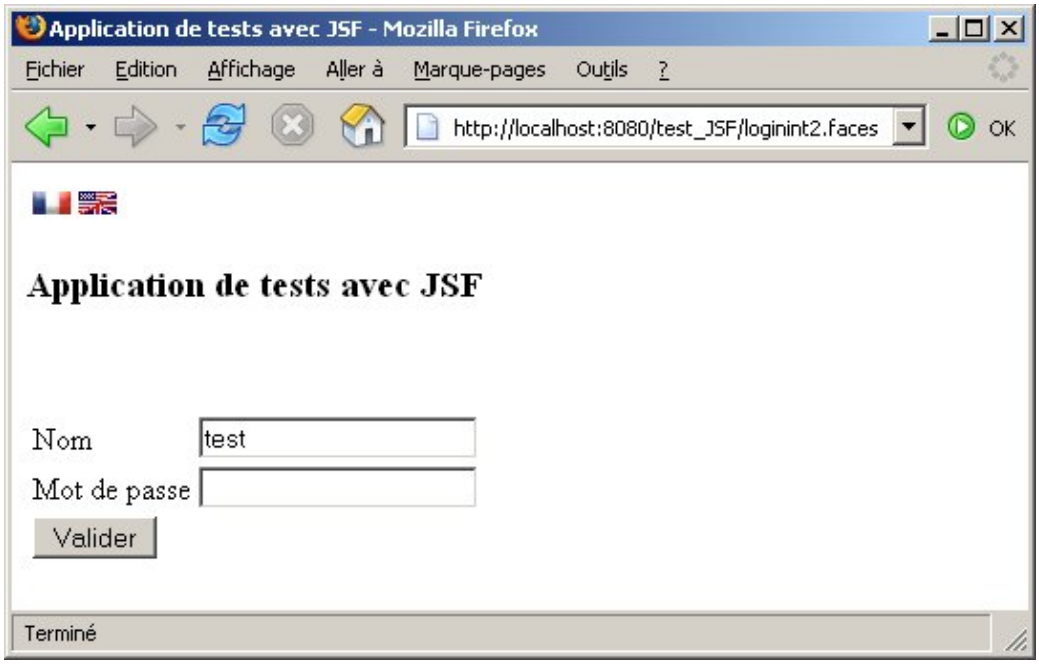
Ce code n'a rien de particulier si ce n'est l'utilisation de l'attribut immediate sur les liens sur le choix de la langue pour empêcher la validation des données lors d'un changement de la langue d'affichage.

Ce sont les deux méthodes du bean qui ce charge de modifier la Locale par défaut du contexte de l'application

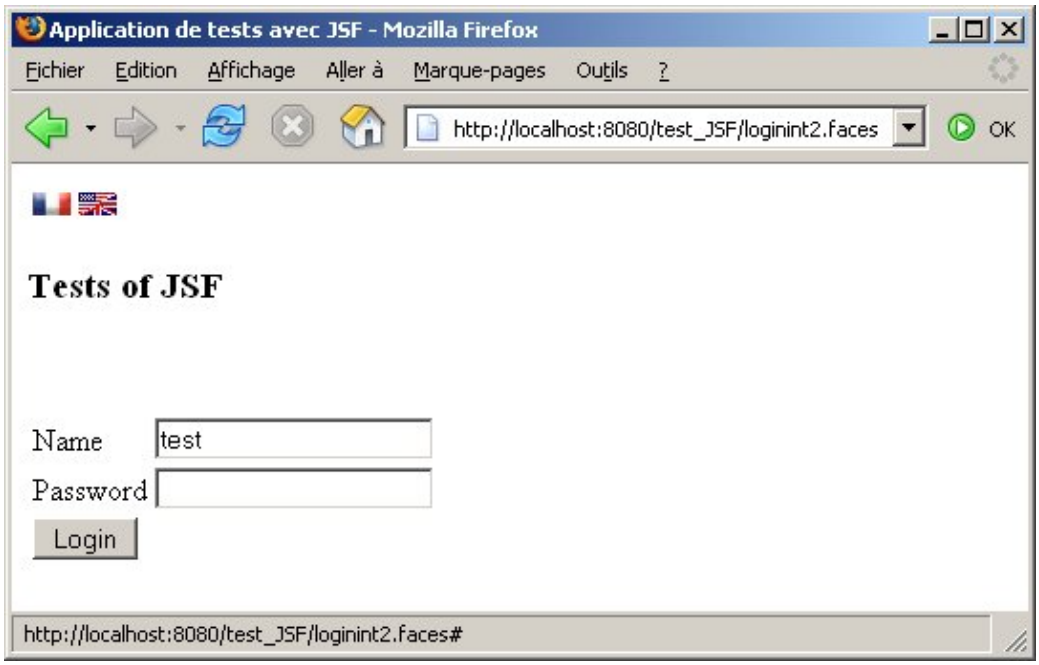
Exemple :

```
package com.jmd.test.jsf;

import java.util.Locale;

import javax.faces.context.FacesContext;

public class LangueApp {

  public String activerFR() {
    FacesContext context = FacesContext.getCurrentInstance();
    context.getViewRoot().setLocale(Locale.FRENCH);
    return null;
  }

  public String activerEN() {
    FacesContext context = FacesContext.getCurrentInstance();
    context.getViewRoot().setLocale(Locale.ENGLISH);
    return null;
  }

}
```

Lors de l'exécution, la page s'affiche en français par défaut.



Lors d'un clic sur la petite icône indicant la langue anglaise, la page est réaffichée en anglais.



39.19. Les points faibles de JSF

Malgré ces nombreux points forts, JSF possède aussi quelques points faibles :

- Maturité de la technologie

  JSF est une technologie récente qui nécessite l'écriture de beaucoup de code. Bien que prévu pour être utilisé dans des outils pour faciliter la rédaction d'une majeure partie de ce code, seul quelques outils supporte JSF.

- Manque de composants évolués en standard

  L'implémentation standard ne propose que des composants simples dont la plupart ont une correspondance directe en HTML. Hormis le composant dataTable aucun composant évolué n'est proposé en standard dans la

version 1.0. Il est donc nécessaire de développer ces propres composants ou d'acquérir les composants nécessaires auprès de tiers.

- Consommation en ressources d'une application JSF

  L'exécution d'une application JSF est assez gourmande en ressource notamment mémoire à cause du mode de fonctionnement du cycle de traitement d'une page. Ce cycle de vie inclus la création en mémoire d'une arborescence des composants de la page utilisée lors des différentes étapes de traitements.

- Le rendu des composants unique en HTML en standard

  Dans l'implémentation de référence le rendu des composants est uniquement possible en HTML, alors que JSF intègre en système de rendu (Renderer) découplé des traitements des composants. Pour un rendu différent de HTML, il est nécessaire développer ces propres Renderers ou d'acquérir un système de rendu auprès de tiers

La suite de ce chapitre sera développée dans une version future de ce document

40. JNDI (Java Naming and Directory Interface)

La suite de ce chapitre sera développée dans une version future de ce document

JNDI est l'acronyme de Java Naming and Directory Interface. Cette API fournit une interface unique pour utiliser différents services de nommages ou d'annuaires:

- LDAP (Lightweigth Directory Access Protocol)
- DNS (Domain Naming Service)
- NIS (Network Information Service) de SUN
- service de nommage CORBA
- service de nommage RMI
- etc ...

Un service de nommage permet d'associer un nom unique à un objet et faciliter ainsi l'obtention de cet objet.

Un annuaire est un service de nommage qui possède en plus une représentation hiérarchique des objets qu'il contient et un mécanisme de recherche.

Pour pouvoir utiliser autant de services différents possédant des protocoles d'accès différents, JNDI utilise des pilotes SPI (Service Provider). Trois de ces pilotes sont fournis en standard :

- LDAP (Lightweigth Directory Access Protocol)
- service de nommage CORBA (COS)
- service de nommage RMI

JNDI est intégré au JDK à partir de sa version 1.3.

JNDI est composé de 6 packages :

Packages	Rôle
javax.naming	Classes et interfaces pour utiliser un service nommage
javax.naming.directory	Classes et interfaces pour utiliser un service d'annuaire
javax.naming.event	Classes et interfaces pour l'émission d'événement lors d'un accès à un service
javax.naming.ldap	Classes et interfaces dédiées pour l'utilisation de LDAP v3
javax.naming.spi	Classes et interfaces dédiées aux Service Provider

Pour plus d'informations sur JNDI : http://java.sun.com/products/jndi

Ce chapitre contient plusieurs sections :

40.1. Les concepts de base

40.1.1. La définition d'un annuaire

Un annuaire est un outil qui permet de stocker et de consulter des informations selon un protocole particulier. Un annuaire est plus particulièrement dédié à la recherche et la lecture d'informations : il est optimisé pour ce type d'activité mais il doit aussi être capable d'ajouter des informations.

40.1.2. Le protocole LDAP

Le protocole LDAP (Ligthweight Directory Access Protocol) a été développé pour être un standard de service d'annuaire.

Un annuaire LDAP peut être utilisé pour :

- l'authentification et le contrôle d'accès des utlisateurs aux applications ou aux ressources.
- l'obtention d'informations sur un élément contenu dans l'annuaire
- l'obtention d'un objet stocké dans l'annuaire

Un annuaire LDAP stocke les informations sous la forme d'une arborescence hiérarchique. Le modèle de cette représentation est stocké dans un schéma. Ceci permet de personnaliser l'arborescence. Le premier élément de l'arborescence est nommé racine.

Chaque élément de l'arborescence est défini par un attribut et une valeur. L'attribut est défini dans le schéma et la valeur est libre. LDAP défini en standard plusieurs attributs.

Pour accéder à un élément particulier, il faut préciser chaque paire attribut/valeur de chaque élément appartenant à l'arborescence pour accéder à l'élément. Cet ensemble de paire attribut/valeur séparée par une virgule est nommé Dn (Distinguished Name)

40.2. Présentation de JNDI

JNDI est un composant important de J2EE car plusieurs technologies comme les EJB utilisent JNDI. JNDI est utilisé pour stocker et obtenir un objet Java. D'autres technologies comme JDBC ou JMS peuvent utiliser JNDI pour les mêmes raisons.

40.3. Utilisation de JNDI avec un serveur LDAP

La première chose à faire est de se connecter au serveur. Il faut utiliser un objet InitialDirContext pour obtenir un objet qui implémente l'interface DirContext. Le constructeur de l'objet InitialDirContext attend un objet de type Hashtable qui contient les paramètres nécessaires à la connexion.

41. JMS (Java Messaging Service)

JMS, acronyme de Java Messaging Service, est une API fournie par Sun pour permettre un dialogue standard entre des applications ou des composants via des brokers de messages ou MOM (Middleware Oriented Messages). Elle permet donc d'utiliser des services de messaging dans des applications java comme le fait l'API JDBC pour les bases de données

Des informations utiles sur JMS peuvent être trouvées à l'URL : http://java.sun.com/products/jms/index.htm

Ce chapitre contient plusieurs sections :

41.1. Présentation de JMS

JMS a été intégré à la plateforme J2EE à partir de la version 1.3 mais il n'existe pas d'implémentation officielle de cette API avant la version 1.3 du J2EE. JMS est utilisable avec les versions antérieures mais elle oblige à utiliser un outil externe qui implémente l'API.

Il existe un certain nombre d'outils qui implémentent JMS dont la majorité sont des produits commerciaux.

Dans la version 1.3 du J2EE, JMS peut être utilisé dans un composant web ou un EJB, un type d'EJB particulier a été ajouté pour traiter les messages et des échanges JMS peuvent être intégrés dans une transaction gérée avec JTA (Java Transaction API).

JMS définit plusieurs entités :

- Un provider JMS : outil qui implémente l'API JMS pour échanger les messages : ce sont les brokers de messages
- Un client JMS : composant écrit en java qui utilise JMS pour émettre et/ou recevoir des messages.
- Un message : données échangées entre les composants

Les messages sont asynchrones mais JMS définit deux modes pour consommer un message :

- Mode synchrone : ce mode nécessite l'appel de la méthode receive() ou d'une de ces surcharges. Dans ce cas, l'application est arrêtée jusqu'à l'arrivée du message. Une version surchargée de cette méthode permet de rendre la main après un certain timeout.
- Mode asynchrone : il faut définir un listener qui va lancer un thread qui va attendre les messages et éxecuter une méthode lors de leur arrivée.

41.2. Les services de messages

Les brokers de messages ou MOM (Middleware Oriented Message) permettent d'assurer l'échange de messages entre deux composants nommés clients. Ces échanges peuvent se faire dans un contexte interne (pour l'EAI) ou un contexte externe (pour le B2B).

Les deux clients n'échangent pas directement des messages : un client envoi un message et le client destinataire doit demander la reception du message. Le transfert du message et sa persistence sont assurés par le broker.

Les échanges de message sont :

- asynchrones :
- fiables : les messages ne sont délivrés qu'un et une seule fois

Les MOM représentent le seul moyen d'effectuer un échange de messages asynchrones. Ils peuvent aussi être très pratiques pour l'échange synchrone de messages plutôt que d'utiliser d'autres méchanismes plus compliqués à mettre en oeuvre (sockets, RMI, CORBA ...).

Les brokers de messages peuvent fonctionner selon deux modes :

- le mode point à point (queue)
- le mode publication/abonnement (publish/souscribe)

Le mode point à point (point to point) repose sur le concept de files d'attente (queues). Le message est stocké dans une file d'attente puis il est lu dans cette file ou dans une autre. Le transfert du message d'une file à l'autre est réalisé par le broker de message.

Chaque message est envoyé dans une seule file d'attente. Il y reste jusqu'à ce qu'il soit consommé par un client et un seul. Le client peut le consommer ultérieurement : la persistance est assurée par le broker de message.

Le mode publication/abonnement repose sur le concept de sujets (Topics). Plusieurs clients peuvent envoyer des messages dans ce topic. Le broker de message assure l'acheminement de ce message à chaque client qui se sera préalablement abonné à ce topic. Le message possède donc potentiellement plusieurs destinataires. L'émetteur du message ne connaît pas les destinataires qui se sont abonnés.

Les principaux brokers de messages commerciaux sont :

Produit	Société	URL
Sonic MQ	Progress software	http://www.progress.com/sonicmq/index.htm
VisiMessage	Borland	http://www.borland.com/appserver
Swift MQ		http://www.swiftmq.com
MessageQ	BEA	http://www.bea.com/products/messageq/datasheet.shtml
MQ Series	IBM	http://www−4.ibm.com/software/ts/mqseries
Rendez vous	Tibco	http://www.tibco.com/products/rv/index.html

Il existe quelques brokers de messages open source :

Outils	URL
OpenJMS	http://openjms.sf.net/
Joram	http://joram.objectweb.org/

41.3. Le package javax.jms

Ce package et ses sous packages contiennent plusieurs interfaces qui définissent l'API.

- Connection
- Session
- Message
- MessageProducer
- MessageListener

41.3.1. La factory de connexion

Un objet factory est un objet qui permet de retourner un objet pour se connecter au broker de messages.

Il faut fournir un certain nombre de paramètres à l'objet factory.

Il existe deux types de factory, QueueConnectionFactory et TopicConnectionFactory selon le type d'échanges que l'on fait. Ce sont des interfaces que le broker de message doit implémenter pour fournir des objets.

Pour obtenir un objet de ce type, il faut soit instancier directement un tel objet soit faire appel à JNDI pour l'obtenir.

41.3.2. L'interface Connection

Cette interface définit des méthodes pour la connection au broker de messages.

Cette connection doit être établie en fonction du mode utilisé :

- l'interface QueueConnection pour le mode point à point
- l'interface TopicConnection pour le mode publication/abonnement

Pour obtenir l'un ou l'autre, il faut utiliser un objet factory correspondant de type QueueConnectionFactory ou TopicConnectionFactory avec la méthode correspondante : createQueueConnection() ou createTopicConnection().

La classe qui implémente cette interface se charge du dialogue avec le broker de message.

La méthode start() permet de démarrer la connexion.

Exemple :

```
connection.start();
```

La méthode stop() permet de suspendre temporairement la connexion.

La méthode close() permet de fermer la connexion.

41.3.3. L'interface Session

Elle représente un contexte transactionnel de réception et d'émission pour une connection donnée.

C'est d'ailleurs à partir d'un objet de type Connection que l'on créé une ou plusieurs sessions.

La session est mono thread : si l'application utilise plusieurs threads qui échangent des messages, il faut définir une session pour chaque thread.

C'est à partir d'un objet session que l'on créé des messages et des objets pour les envoyer et les recevoir.

Comme pour la connexion, la création d'un objet de type Session dépend du mode de fonctionnement. L'interface Session possède deux interfaces filles :

- l'interface QueueSession pour le mode point à point
- l'interface TopicSession pour le mode publication/abonnement

Pour obtenir l'un ou l'autre, il faut utiliser un objet Connection correspondant de type QueueConnection ou TopicConnection avec la méthode correspondante : createQueueSession() ou createTopicSession().

Ces deux méthodes demandent deux paramètres : un boolean qui indique si la session gère une transaction, et une constante qui précise le mode d'accusé de reception des messages.

Il existe trois modes d'accusés de reception (trois constantes sont définies dans l'interface Session) :

- AUTO_ACKNOWLEDGE : l'accusé de reception est automatique
- CLIENT_ACKNOWLEDGE : c'est le client qui envoi l'accusé grace à l'appel de la méthode acknowledge() du message
- DUPS_OK_ACKNOWLEDGE : ce mode permet de dupliquer un message

L'interface Session définit plusieurs méthodes dont les principales:

Méthode	Rôle
void close()	fermer la session
void commit()	valider la transaction
XXX createXXX()	permet de créer un Message dont le type est XXX
void rollback()	Invalide la transaction

41.3.4. Les messages

Ils doivent obligatoirement implémenter l'interface Message ou l'une de ces sous classes.

Les messages sont composés de trois parties :

- l'en tête (header)
- les propriétés (properties)
- le corps du message (body)

41.3.4.1. L'en tête

Cette partie du message contient un certain nombre de champs prédéfinis qui contiennent des données pour identifier et acheminer le message.

La plupart de ces données sont renseignées lors de l'appel à la méthode send() ou publish().

Les champs les plus importants sont :

Nom	Rôle
JMSMessageID	identifiant unique du message
JMSDestination	file d'attente ou topic destinataire du message
JMSCorrelationID	utilisé pour synchroniser de façon applicative deux messages de la forme requête/réponse. Dans ce cas, dans le message réponse, ce champ contient le messageID du message requête

41.3.4.2. Les propriétés

Ce sont des champs supplémentaires : certains sont définis par JMS mais il est possible d'ajouter ces propres champs.

Cette partie du message est optionnelle.

Elles permettent de définir des données qui seront utilisées pour fournir des données supplémentaires ou pour filtrer le message.

41.3.4.3. Le corps du message

Il contient les données du message : ils sont formattés selon le type du message.

Cette partie du message est optionnelle.

Les messages peuvent être de plusieurs types, définis dans les interfaces suivantes :

type	Interface	Role
bytes	BytesMessage	échange d'octets
texte	TextMessage	échange de données texte (XML par exemple)
object	ObjectMessage	échange d'objets Java qui doivent être sérialisables
Map	MapMessage	échange de données sous la forme clé/valeur. La clé doit être une chaîne de caractères et la valeur de type primitive
Stream	StreamMessage	échange de données en provenance d'un flux

Il est possible de définir son propre type qui doit obligatoirement implémenter l'interface Message.

C'est un objet de type Session qui contient les méthodes nécessaires à la création d'un message selon son type.

Lors de la réception d'un message, celui ci est toujours de type Message : il faut effectuer un transtypage en fonction de son type en utilisant l'opérateur instanceof. A ce moment, il faut utiliser le getter correspondant pour obtenir les données.

Exemple :

```
Message message = ...

 if (message instanceof TextMessage) {
    TextMessage textMessage = (TextMessage) message;
    System.out.println("message: " + textMessage.getText());
 }
```

41.3.5. L'envoi de Message

L'interface MessageProducer est la super interface des interfaces qui définissent des méthodes pour l'envoi de messages.

Il exite deux interfaces filles selon le mode de fonctionnement pour envoyer un message : QueueSender et TopicPublisher.

Ces objets sont créés à partir d'un objet représentant la session :

- la méthode createSender() pour obtenir un objet de type QueueSender
- la méthode createPublisher() pour obtenir un objet de type TopicPublisher

Ces objets peuvent être liés à une entité physique par exemple une file d'attente particulière pour un objet de type QueueSender. Si ce n'est pas le cas, cette entité devra être précisée lors de l'envoie du message en utilisant une version surchargée de la méthode chargée de l'emission du message.

41.3.6. La réception de messages

L'interface MessageConsumer est la super interface des interfaces qui définissent des méthodes pour la réception de messages.

Il existe des interfaces selon le mode fonctionnement pour recevoir un message QueueReceiver et TopicSubscriber.

La réception d'un message peut se faire avec deux modes :

- synchrone : dans ce cas, l'attente d'un message bloque l'execution du reste de code
- asynchrone : dans ce cas, un thread est lancé qui attend le message et appelle une méthode (callback) à son arrivée. L'execution de l'application n'est pas bloquée.

L'interface MessageConsumer définit plusieurs méthodes sont les principales sont :

Méthode	Rôle
close()	fermer l'objet qui reçoit les messages pour le rendre inactif
Message receive()	attend et retourne le message à son arrivée
Message receice(long)	attend durant le nombre de milliseconde précisé en paramètre et renvoie le message si il arrive durant ce laps de temps
Message receiveNoWait()	renvoie un message sans attendre si il y en a un de présent
setMessageListener(MessageListener)	associe un Listener pour traiter les messages de façon asynchrone

Pour obtenir un objet qui implémente l'interface QueueReceiver, il faut utiliser la méthode createReceiver() d'un objet de type QueueSession.

Pour obtenir un objet qui implémente l'interface TopicSubscriber, il faut utiliser la méthode createSubscriber() d'un objet de type TopicSession.

41.4. L'utilisation du mode point à point (queue)

41.4.1. La création d'une factory de connexion : QueueConnectionFactory

Un objet factory est un objet qui permet de retourner un objet pour se connecter au broker de messages.

Pour obtenir un objet de ce type, il faut soit instancier directement un tel objet soit faire appel à JNDI pour l'obtenir.

Exemple avec MQSeries :

```
String qManager = ...
String hostName = ...
String channel = ...

MQQueueConnectionFactory factory = new MQQueueConnectionFactory();
factory.setQueueManager(qManager);
factory.setHostName(hostName);
factory.setChannel(channel);
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
```

41.4.2. L'interface QueueConnection

Cette interface hérite de l'interface Connection.

Pour obtenir un objet qui implémente cette interface, il faut utiliser un objet factory correspondant de type QueueConnectionFactory avec la méthode correspondante : createQueueConnection().

Exemple :
`QueueConnection connection = factory.createQueueConnection();` `connection.start();`

L'interface QueueConnection définit plusieurs méthodes dont la principale est :

Méthode	Rôle
QueueSession createQueueSession(boolean, int)	renvoie un objet qui définit la session. Le boolean précise si la session gère une transaction. L'entier précise le mode d'accusé de reception.

41.4.3. La session : l'interface QueueSession

Elle hérite de l'interface Session.

Pour obtenir un objet qui implémente cette interface, il faut utiliser la méthode createQueueSession() d'un objet connexion de type QueueConnection.

Exemple :
`QueueSession session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);`

L'interface QueueSession définit plusieurs méthodes dont les principales sont :

Méthode	Rôle
QueueReceiver createQueueReceiver(Queue)	renvoie un objet qui définit une file d'attente de réception
QueueSender createQueueSender(Queue)	renvoie un objet qui définit une file d'attente d'émission

41.4.4. L'interface Queue

Un objet qui implémente cette interface encapsule une file d'attente particulière.

Pour obtenir un objet qui implémente cette interface, il faut utiliser la méthode createQueue() d'un objet de type QueueSession.

Exemple avec MQseries :
`Queue fileEnvoi =` `session.createQueue("queue:///file.out"?expiry=0&persistence=1&targetClient=1");`

41.4.5. La création d'un message

Pour créer un message, il faut utiliser une méthode createXXXMessage() d'un objet QueueSession ou XXX représente le type du message.

Exemple :

```
String message = «bonjour»;
TextMessage textMessage = session.createTextMessage();
textMessage.setText(message);
```

41.4.6. L'envoi de messages : l'interface QueueSender

Cette interface hérite de l'interface MessageProducer.

Pour obtenir un objet qui implémente cette interface, il faut utiliser la méthode createQueueSender() d'un objet de type QueueSession.

Exemple :

```
QueueSender queueSender = session.createSender(fileEnvoi);
```

Il est possible de fournir un objet de type Queue qui représente la file d'attente : dans ce cas, l'objet QueueSender est lié à cette file d'attente. Si l'on ne précise pas de file d'attente (null fourni en paramètre), dans ce cas, il faudra obligatoirement utiliser une version surchargée de la méthode send() lors de l'envoi pour préciser la file d'attente.

Avec un objet de type QueueSender, la méthode send() permet l'envoie d'un message dans la file d'attente. Cette méthode possède plusieurs surcharges :

Méthode	Rôle
void send(Message)	Envoie le message dans la file d'attente définit dans l'objet de type QueueSender
void send(Queue, Message)	Envoie le message dans la file d'attente fournie en paramètre

Exemple :

```
queueSender.send(textMessage);
```

41.4.7. La réception de messages : l'interface QueueReceiver

Cette interface hérite de l'interface MessageConsummer.

Pour obtenir un objet qui implémente cette interface, il faut utiliser la méthode createQueueReceiver() à partir d'un objet de type QueueSession.

Exemple :

```
QueueReceiver queueReceiver = session.createReceiver(fileReception);
```

Il est possible de fournir un objet de type Queue qui représente la file d'attente : dans ce cas, l'objet QueueSender est lié à cette file d'attente. Si l'on ne précise pas de file d'attente (null fourni en paramètre), dans ce cas, il faudra obligatoirement utiliser une version surchargée de la méthode receive() lors de l'envoi pour préciser la file d'attente.

Cette interface ne définit qu'une seule méthode supplémentaire :

Méthode	Rôle
Queue getQueue()	renvoie la file d'attente associée à l'objet

La réception de messages peut se faire dans le mode synchrone ou asynchrone.

41.4.7.1. La réception dans le mode synchrone

Dans ce mode, le programme est interrompu jusqu'à l'arrivée d'un nouveau message. Il faut utiliser la méthode receive() héritée de l'interface MessageConsumer. Il existe plusieurs méthodes et surcharges de ces méthodes qui permettent de faire face à toute les utilisations :

- receiveNoWait() : renvoi un message présent sans attendre
- receive(long) : renvoi un message qui arrive durant le temps fourni en paramètre
- receive() : renvoi le message dès qu'il arrive

Exemple :

```
Message message = null;
message = queueReceiver.receive(10000);
```

41.4.7.2. La réception dans le mode asynchrone

Dans ce mode, le programme n'est pas interrompu mais un objet écouteur va être enregistré auprès de l'objet de type QueueReceiver. Cet objet qui implémente l'interface MessageListener va être utilisé comme gestionnaire d'évenements lors de l'arrivé d'un nouveau message.

L'interface MessageListener ne définit qu'une seule méthode qui reçoit en paramètre le message : onMessage(). C'est cette méthode qui sera appelée lors de la reception d'un message.

41.4.7.3. La sélection de messages

Une version surchargée de la méthode createReceiver() d'un objet de type QueueSession permet de préciser dans ces paramètres une chaîne de caractères qui va servir de filtre sur les messages à recevoir.

Dans ce cas, le filtre est effectué par le broker de message plutôt que par le programme.

Cette chaîne de caractères contient une expression qui doit avoir une syntaxe proche d'une condition SQL. Les critères de la sélection doivent porter sur des champs inclus dans l'en tête ou dans les propriétés du message. Il n'est pas possible d'utiliser des données du corps du message pour effectuer le filtre.

Exemple : envoi d'un message requête et attente de sa réponse. Dans ce cas, le champ JMSCorrelationID du message réponse contient le JMSMessageID du message requete

```
String messageEnvoie = «bonjour»;
TextMessage textMessage = session.createTextMessage();
textMessage.setText(messageEnvoie);
queueSender.send(textMessage);
int correlId = textMessage.getJMSMessageID();
QueueReceiver queueReceiver = session.createReceiver(
fileEnvoie, "JMSCorrelationID = '" + correlId +"'");
Message message = null;
message = queueReceiver.receive(10000);
```

41.5. L'utilisation du mode publication/abonnement (publish/souscribe)

41.5.1. La création d'une factory de connexion : TopicConnectionFactory

Un objet factory est un objet qui permet de retourner un objet pour se connecter au broker de messages.

Pour obtenir un objet de ce type, il faut soit instancier directement un tel objet soit faire appel à JNDI pour l'obtenir.

41.5.2. L'interface TopicConnection

Cette interface hérite de l'interface Connection.

Pour obtenir un objet qui implémente cette interface, il faut utiliser un objet factory correspondant de type TopicConnectionFactory avec la méthode correspondante : createTopicConnection().

Exemple :

```
TopicConnection connection = factory.createTopicConnection();
connection.start();
```

L'interface TopicConnection définit plusieurs méthodes dont la principale est :

Méthode	Rôle
TopicSession createTopicSession(boolean, int)	renvoie un objet qui définit la session. Le boolean précise si la session gère une transaction. L'entier précise le mode d'accusé de reception.

41.5.3. La session : l'interface TopicSession

Elle hérite de l'interface Session.

Pour obtenir un objet qui implémente cette interface, il faut utiliser la méthode createTopicSession() d'un objet connexion de type TopicConnection.

Exemple :

```
TopicSession session = connection.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
```

L'interface TopicSession définit plusieurs méthodes dont les principales sont :

Méthode	Rôle
TopicSubscriber createSubscriber(Topic)	renvoie un objet qui permet l'envoie de messages dans un topic
TopicPublisher createPublisher(Topic)	renvoie un objet qui permet la reception de messages dans un topic
Topic createTopic(String)	création d'un topic correspondant à la désignation fournie en paramètre

41.5.4. L'interface Topic

Un objet qui implémente cette interface encapsule un sujet.

Pour obtenir un objet qui implémente cette interface, il faut utiliser la méthode createTopic() d'un objet de type TopicSession.

41.5.5. La création d'un message

Pour créer un message, il faut utiliser une méthode createXXXMessage() d'un objet TopicSession ou XXX représente le type du message.

Exemple :

```
String message = «bonjour»;
TextMessage textMessage = session.createTextMessage();
textMessage.setText(message);
```

41.5.6. L'émission de messages : l'interface TopicPublisher

Cette interface hérite de l'interface MessageProducer.

Avec un objet de type TopicPublisher, la méthode publish() permet l'envoie du message. Cette méthode possède plusieurs surcharges :

Méthode	Rôle
void publish(Message)	envoie le message dans le topic défini dans l'objet de type TopicPublisher
void publish(Topic, Message)	envoie le message dans le topic fourni en paramètre

41.5.7. La réception de messages : l'interface TopicSubscriber

Cette interface hérite de l'interface MessageProducer.

Pour obtenir un objet qui implémente de cette interface, il faut utiliser la méthode createSubscriber() à partir d'un objet de type TopicSession.

Exemple :

```
TopicSubscriber topicSubscriber = session.createSubscriber(topic);
```

Il est possible de fournir un objet de type Topic qui représente le topic : dans ce cas, l'objet Topicsubscriber est lié à ce topic. Si l'on ne précise pas de topic (null fourni en paramètre), dans ce cas, il faudra obligatoirement utilisée un version surchargé de la méthode receive() lors de l'envoi pour préciser le topic.

Cette interface ne définit qu'une seule méthode supplémentaire :

Méthode	Rôle
Topic getTopic()	renvoie le topic associée à l'objet

41.6. Les exceptions de JMS

Plusieurs exceptions sont définies par l'API JMS. La classe mère de toute ces exceptions est la classe JMSException.

Les exceptions définies sont : IllegalStateException, InvalidClientIDException, InvalidDestinationException, InvalidSelectorException, JMSSecurityException, MessageEOFException, MessageFormatException, MessageNotReadableException, MessageNotWriteableException, ResourceAllocationException, TransactionInProgressException,TransactionRolledBackException

La méthode getErrorCode() permet d'obtenir le code erreur spécifique du produit sous forme de chaîne de caractères.

La suite de ce chapitre sera développée dans une version future de ce document

42. JavaMail

Le courrier électronique repose sur le concept du client/serveur. Ainsi, l'utilisation d'e mail requiert deux composants :

- un client de mail (Mail User Agent : MUA) tel que Outlook, Messenger, Eudora ...
- un serveur de mail (Mail Transport Agent : MTA) tel que SendMail

Les clients de mail s'appuient sur un serveur de mail pour obtenir et envoyer des messages. Les échanges entre client et serveur sont normalisés par des protocoles particuliers.

JavaMail est une API qui permet d'utiliser le courrier électronique (e−mail) dans une application écrite en java (application cliente, applet, servlet, EJB ...). Son but est d'être facile à utiliser, de fournir une souplesse qui permette de la faire évoluer et de rester le plus indépendant possible des protocoles utilisés.

JavaMail est une extension au JDK qui n'est donc pas fournie avec J2SE. Pour l'utiliser, il est possible de la télécharger sur le site de SUN : http://java.sun.com/products/javamail. Elle est intégré au J2EE.

Les classes et interfaces sont regroupées dans quatre packages : javax.mail, javax.mail.event, javax.mail.internet, javax.mail.search.

Il existe deux versions de cette API :

- 1.1.3 : version fournie avec J2EE 1.2
- 1.2 : version courante

Les deux versions fonctionnent avec un JDK dont la version est au moins 1.1.6.

Cette API permet une abstraction assez forte de tout système de mail, ce qui lui permet d'ajouter des protocoles non gérés en standard. Pour gérer ces différents protocoles, il faut utiliser une implémentation particulière pour chacun d'eux, fournis par des fournisseurs tiers. En standard, JavaMail 1.2 fournie une implémentation pour les protocoles SMTP, POP3 et IMAP4. JavaMail 1.1.3 ne fournie une implémentation que pour les protocoles SMTP et IMAP : l'implémentation pour le protocole POP3 doit être téléchargée séparément.

Ce chapitre contient plusieurs sections :

42.1. Téléchargement et installation

Pour le J2SE, il est nécessaire de télécharger les fichiers utiles et de les installer.

Pour les deux versions de l'API, il faut télécharger la version correspondante, dezipper le fichier dans un répertoire et

ajouter le fichier mail.jar dans le CLASSPATH.

Ensuite il faut aussi installer le framework JAF (Java Activation Framework) : télécharger le fichier, dezipper et ajouter le fichier activation.jar dans le CLASSPATH

Pour pouvoir utiliser le protocole POP3 avec JavaMail 1.1.3, il faut télécharger en plus l'implémentation de ce protocole et inclure le fichier POP3.jar dans le CLASSPATH.

Pour le J2EE 1.2.1, l'API version 1.1.3 est intégrée à la plate−forme. Elle ne contient donc pas l'implémentation pour le protocole POP3. Il faut la télécharger et l'installer en plus comme avec le J2SE.

Pour le J2EE 1.3, il n'y a rien de particulier à faire puisque l'API version 1.2 est intégrée à la plate−forme.

42.2. Les principaux protocoles

42.2.1. SMTP

SMTP est l'acronyme de Simple Mail Transport Protocol. Ce protocole défini par la recommandation RFC 821 permet l'envoi de mails vers un serveur de mails qui supporte ce protocole.

42.2.2. POP

POP est l'acronyme de Post Office Protocol. Ce protocole défini par la recommandation RFC 1939 permet la réception de mail à partir d'un serveur de mail qui supporte ce protocole. La version courante de ce protocole est 3. C'est un protocole très populaire sur Internet. Il définit une boite au lettre unique pour chaque utilisateur. Une fois que le message est reçu par le client, il est effacé du serveur.

42.2.3. IMAP

IMAP est l'acronyme de Internet Message Acces Procol. Ce protocole défini par la recommandation RFC 2060 permet aussi la réception de mail à partir d'un serveur de mail qui supporte ce protocole. La version courante de ce protocole est 4. Ce protocole est plus complexe car il apporte des fonctionnalités supplémentaires : plusieurs répertoires par utilisateur, partage de répertoire entre plusieurs utilisateurs, maintient des messages sur le serveur , etc ...

42.2.4. NNTP

NNTP est l'acronyme de Network News Transport Protocol. Ce protocole est utilisé par les forums de discussion (news).

42.3. Les principales classes et interfaces de l'API JavaMail

JavaMail propose des classes et interfaces qui encapsulent ou définissent les objets liés à l'utilisation des mails et les protocoles utilisés pour les échanger.

42.3.1. La classe Session

La classe Session encapsule pour un client donné sa connexion avec le serveur de mail. Cette classe encapsule les données liées à la connexion (options de configuration et données d'authentification). C'est à partir de cet objet que toutes les actions concernant les mails sont réalisées.

Les paramètres nécessaires sont fournis dans un objet de type Properties. Un objet de ce type est utilisé pour contenir les variables d'environnements : placer certaines informations dans cet objet permet de partager des données.

Une session peut être unique ou partagée par plusieurs entités.

Exemple :

```
// creation d'une session unique
 Session session = Session.getInstance(props,authenticator);
 // creation d'une session partagee
 Session defaultSession = Session.getDefaultInstance(props,authenticator);
```

Pour obtenir une session, deux paramètres sont attendus :

- un objet Properties qui contient les paramètres d'initialisation. Un tel objet est obligatoire
- un objet Authenticator optionnel qui permet d'authentifier l'utilisateur auprès du serveur de mail

La méthode setDebug() qui attend en paramètre un booleen est très pratique pour debugger car avec le paramètre true, elle affiche des informations lors de l'utilisation de la session notamment le détail des commandes envoyées au serveur de mail.

42.3.2. Les classes Address, InternetAddress et NewsAddress

La classe Address est une classe abstraite dont héritent toutes les classes qui encapsulent une adresse dans un message.

Deux classes filles sont actuellement définies :

- InternetAddress
- NewsAddress

Le classe InternetAddress encapsule une adresse email respectant le format de la RFC 822. Elle contient deux champs : address qui contient l'adresse e mail et personal qui contient le nom de la personne. La classe possède des constructeurs, des getters et des setters pour utiliser ces attributs.

Le plus simple pour créer un objet InternetAddress est d'appeler le constructeur en lui passant en paramètre une chaîne de caractère contenant l'adresse e−mail.

Exemple :

```
InternetAddress vInternetAddresses = new InternetAddress;
 vInternetAddresses = new InternetAddress("moi@chez-moi.fr");
```

Un second constructeur permet de préciser l'adresse e−mail et un nom en clair.

La méthode getLocalAddress(Session) permet de déterminer si possible l'objet InternetAddress encapsulant l'adresse e mail de l'utilisateur courant, sinon elle renvoie null.

La méthode parse(String) permet de créer un tableau d'objet InternetAddress à partir d'une chaîne contenant les adresses e mail séparées par des virgules.

Un objet InternetAddress est nécessaire pour chaque émetteur et destinataire du mail. L'API ne vérifie pas l'existence des adresses fournies. C'est le serveur de mail qui vérifiera les destinataires et éventuellement les émetteurs selon son

paramétrage.

La classe NewsAddress encapsule une adresse news (forum de discussion) respectant le format RFC1036. Elle contient deux champs : host qui contient le nom du serveur et newsgroup qui le nom du forum

La classe possède des constructeurs, des getters et des setters pour utiliser ces attributs.

42.3.3. L'interface Part

Cette interface définit un certain nombre d'attributs commun à la plupart des systèmes de mail et un contenu.

Le contenu peut être renvoyé sous trois formes : DataHandler, InputStream et Object.

Cette interface définit plusieurs méthodes principalement des getters et des setters dont les principaux sont :

Méthode	Rôle
int getSize()	Renvoie la taille du contenu sinon −1 si elle ne peut être déterminée
int getLineCount()	Renvoie le nombre de ligne du contenu sinon −1 s'il ne peut être déterminé
String getContentType()	Renvoie le type du contenu sinon null
String getDescription()	Renvoie la description
void setDescription(String)	Mettre à jour la description
InputStream getInputStream()	Renvoie le contenu sous le forme d'un flux
DataHandler getDataHandler()	Renvoie le contenu sous la forme d'un objet DataHandler
Object getContent()	Renvoie le contenu sous la forme d'un objet. Un cast est nécessaire selon le type du contenu.
void setText(String)	Mettre à jour le contenu sous forme d'une chaîne de caractères fournie en paramètre

42.3.4. La classe Message

La classe abstraite Message encapsule un Message. Le message est composé de deux parties :

- une en−tête qui contient des attributs
- un corps qui contient les données à envoyer

Pour la plupart de ces données, la classe Message implémente l'interface Part qui encapsule les attributs nécessaires à la distribution du message (auteur, destinataire, sujet ...) et le corps du message.

Le contenu du message est stocké sous forme d'octet. Pour accéder à son contenu, il faut utiliser un objet du JavaBean Activation Framework (JAF) : DataHandler. Ceci permet une séparation des données nécessaires à la transmission et du contenu du message qui peut ainsi prendre n'importe quel format. La classe Message ne connaît pas directement le type du contenu du corps du message.

JavaMail fourni en standard une classe fille nommée MimeMessage qui implémente la recommandation RFC 822 pour les messages possédant un type Mime.

Il y a deux façons d'obtenir un objet de type Message : instancier une classe fille pour créer un nouveau message ou utiliser un objet de type Folder pour obtenir un message existant.

La classe Message défini deux constructeurs en plus du constructeur par défaut :

Constructeur	Rôle
Message(session)	Créer un nouveau message
Message(Folder, int)	Créer un message à partir d'un message existant

La classe MimeMessage est la seule classe fille qui hérite de la classe Message. Elle dispose de plusieurs constructeurs.

Exemple :

```
MimeMessage message = new MimeMessage(session);
```

Elle possèdent de nombreuses méthodes pour initialiser les données du message :

Méthode	Rôle
void addFrom(Address[])	Ajouter des émetteurs au message
void addRecipient(RecepientType, Address[])	Ajouter des destinataires à un type (direct, en copie ou en copie cachée)
Flags getFlags()	Renvoie les états du message
Adress[] getFrom()	Renvoie les émetteurs
int getLineCount()	Renvoie le nombre ligne du message
Address[] getRecipients(RecepientType)	Renvoie les destinataires du type fourni en paramètre
Address getReplyTo()	Renvoie les email pour la réponse
int getSize()	Renvoie la taille du message
String getSubject()	Renvoie le sujet
Message reply(boolean)	Créer un message pour la réponse : le booleen indique si la réponse ne doit être faite qu'à l'émetteur
void set content(Object, String)	Mettre à jour le contenu du message en précisant son type mime
void seFrom(Address)	Mettre à jour l'émetteur
void setRecipients(RecepientsType, Address[])	Mettre à jour les destinataires d'un type
void setSendDate(Date)	Mettre à jour la date d'envoi
void setText(String)	Mettre à jour le contenu du message avec le type mime « text/plain »
void setReply(Address)	Mettre à jour le destinataire de la réponse
void writeTo(OutputStream)	Envoie le message au format RFC 822 dans un flux. Très pratique pour visualiser le message sur la console en passant en paramètre (System.out)

La méthode addRecipient() permet d'ajouter un destinataire et le type d'envoi.

Le type d'envoi est précisé grâce une constante pour chaque type :

- destinataire direct : Message.RecipientType.TO
- copie conforme : Message.RecipientType.CC
- copie cachée : Message.RecipientType.BCC

La méthode setText() permet de facilement mettre une chaîne de caractères dans le corps du message avec un type MIME « text/plain ». Pour envoyer un message dans un format différent, par exemple HTML, il utilise la méthode setContent() qui attend en paramètre un objet et un chaîne qui contient le type MIME du message.

Exemple :

```
String texte = "<H1>bonjour</H1><a
     href=\"mailto:moi@moi.fr\">mail</a>";
 message.setContent(texte, "text/html");
```

Il est possible de joindre avec le mail des ressources sous forme de pièces jointes (attachments). Pour cela, il faut :

- instancier un objet de type MimeMessage
- renseigner les éléments qui composent l'en−tête : émetteur, destinataire, sujet ...
- Instancier un objet de type MimeMultiPart
- Instancier un objet de type MimeBodyPart et alimenter le contenu de l'élément
- Ajouter cet objet à l'objet MimeMultiPart grâce à la méthode addBodyPart()
- Répéter l'instanciation et l'alimentation pour chaque ressource à ajouter
- utiliser la méthode setContent() du message en passant en paramètre l'objet MimeMultiPart pour associer le message et les pièces jointes au mail

Exemple :

```
Multipart multipart = new MimeMultipart();

 // creation partie principale du message
 BodyPart messageBodyPart = new MimeBodyPart();
 messageBodyPart.setText("Test");
 multipart.addBodyPart(messageBodyPart);

 // creation et ajout de la piece jointe
 messageBodyPart = new MimeBodyPart();
 DataSource source = new FileDataSource("image.gif");
 messageBodyPart.setDataHandler(new DataHandler(source));
 messageBodyPart.setFileName("image.gif");
 multipart.addBodyPart(messageBodyPart);

 // ajout des éléments au mail
 message.setContent(multipart);
```

42.3.5. Les classes Flags et Flag

Cette classe encapsule un ensemble d'états pour un message.

Il existe deux type d'états : les états prédéfinis (System Flag) et les états particuliers définis par l'utilisateur (User Defined Flag)

Un état prédéfini est encapsulé par la classe Internet Flags.Flag. Cette classe définit plusieurs états statiques :

Etat	Rôle
Flags.Flag.ANSWERED	Le message a été demandé : positionné par le client
Flags.Flag.DELETED	Le message est marqué pour la suppression
Flags.Flag.DRAFT	Le message est un brouillon
Flags.Flag.FLAGGED	Le message est marqué dans un état qui n'a pas de définition particulière
Flags.Flag.RECENT	Le message est arrivé recemment. Le client ne peut pas modifier cet état.

Flags.Flag.SEEN	Le message a été visualisé : positionné à l'ouverture du message
Flags.Flag.USER	Le client a la possibilité d'ajouter des états particuliers

Tous ces états ne sont pas obligatoirement supportés par le serveur.

La classe Message possède plusieurs méthodes pour gérer les états d'un message. La méthode getFlags() renvoie un objet Flag qui contient les états du message. Les méthodes setFlag(Flag, boolean) permettent d'ajouter un état du message. La méthode contains(Flag) vérifie si l'état fourni en paramètre est positionné pour le message.

La classe Flags possède plusieurs méthodes pour gérer les états dont les principales sont :

Méthode	Rôle
void add(Flags.Flag)	Permet d'ajouter un état
void add(Flags)	Permet d'ajouter un ensemble d'état
void remove(Flags.Flag)	Permet d'enlever un état
void remove(Flags)	Permet d'enlever un ensemble d'état
boolean contains(Flags.Flag)	Permet de savoir si un état est positionné

42.3.6. La classe Transport

La classe Transport se charge de réaliser l'envoi du message avec le protocole adéquat. C'est une classe abstraite qui contient la méthode static send() pour envoyer un mail.

Il est possible d'obtenir un objet Transport dédié au protocole particulier utilisé par la session en utilisant la méthode getTransport() d'un objet Session. Dans ce cas, il faut :

1. établir la connexion en utilisant la méthode connect() avec le nom du serveur, le nom de l'utilisateur et son mot de passe
2. envoyer le message en utilisant la méthode sendMessage() avec le message et les destinataires. La méthode getAllRecipients() de la classe message permet d'obtenir ceux contenus dans le message.
3. fermer la connexion en utilisant la méthode close()

Il est préférable d'utiliser une instance de Transport tel qu'expliqué ci dessus lorsqu'il y a plusieurs mails à envoyer car on peut maintenir la connexion avec le serveur ouverte pendant les envois.

La méthode static send() ouvre et ferme la connexion à chacun de ces appels.

42.3.7. La classe Store

La classe abstraite store qui représente un système de stockage de messages. Pour obtenir une instance de cette classe, il faut utiliser la méthode getStore() d'un objet de type Session en lui donnant comme paramètre le protocole utilisé.

Pour pouvoir dialoguer avec le serveur de mail, il faut appeler la méthode connect() en lui précisant le nom du serveur, le nom d'utilisateur et le mot de passe de l'utilisateur.

La méthode close() permet de libérer la connexion avec le serveur.

42.3.8. La classe Folder

La classe abstraite Folder représente un répertoire dans lequel les messages sont stockés. Pour obtenir un instance de cette classe, il faut utiliser la méthode getFolder() d'un objet de type Store en lui précisant le nom du répertoire.

Avec le protocole POP3 qui ne gère qu'un seul répertoire, le seul possible est « INBOX ».

Pour pouvoir être utilisé, il faut appeler la méthode open() de la classe Folder en lui précisant le mode d'utilisation : READ_ONLY ou READ_WRITE.

Pour obtenir les messages contenus dans le répertoire, il faut appeler la méthode getMessages(). Cette méthode renvoie un tableau de Message qui peut être null si aucun message n'est renvoyé.

Une fois les opérations terminées, il faut fermer le répertoire en utilisant la méthode close().

42.3.9. Les propriétés d'environnement

JavaMail utilise des propriétés d'environnement pour recevoir certains paramètres de configuration. Ils sont stockés dans un objet de type Properties.

L'objet Properties peut contenir un certains nombre de propriétés qui possèdent des valeurs par défaut :

Propriété	Rôle	Valeur par défaut
mail.store.protocol	Protocole de stockage du message	le premier protocole concerné dans le fichier de configuration
mail.transport.protocol	Protocole de transport par défaut	le premier protocole concerné dans le fichier de configuration
mail.host	Serveur de Mail par défaut	localhost
mail.user	Nom de l'utilisateur pour se connecter au serveur de mail	user.name
mail.protocol.host	Serveur de mail pour un protocole dédié	mail.host
mail.protocol.user	Nom de l'utilisateur pour se connecter au serveur de mail pour un protocole dédié	
mail.from	adresse par défaut de l'expéditeur	user.name@host
mail.debug	mode de debuggage par défaut	

Attention : l'utilisation de JavaMail dans une applet implique de fournir explicitement toutes les valeurs des propriétés utiles car une applet n'a pas la possibilité de définir toutes les valeurs par défaut car l'accès à ces propriétés est restreint.

L'usage de certains serveurs de mail nécessite l'utilisation d'autres propriétés.

42.3.10. La classe Authenticator

Authenticator est une classe abstraite qui propose des méthodes de base pour permettre d'authentifier un utilisateur. Pour l'utiliser, il faut créer une classe fille qui se chargera de collecter les informations. Plusieurs méthodes appelées selon les besoins sont à redéfinir :

Méthode	Rôle

String getDefaultUserName()	
PasswordAuthentication getPasswordAuthentication()	
int getRequestingPort()	
String getRequestingPort()	
String getRequestingProtocol()	
InetAddress getRequestingSite()	

Par défaut, la méthode getPasswordAuthentication() de la classe Authentication renvoie null. Cette méthode renvoie un objet PasswordAuthentication à partir d'une source de données (boîte de dialogue pour saisie, base de données ...).

Une instance d'une classe fille de la classe Authenticator peut être fournie à la session. L'appel à Authenticator sera fait selon les besoins par la session.

42.4. L'envoie d'un e mail par SMTP

Pour envoyer un e mail via SMTP, il faut suivre les principales étapes suivantes :

- Positionner les variables d'environnement nécessaires
- Instancier un objet Session
- Instancier un objet Message
- Mettre à jour les attributs utiles du message
- Appeler la méthode send() de la classe Transport

Exemple :

```
import javax.mail.internet.*;
import javax.mail.*;
import java.util.*;

/**
 * Classe permettant d'envoyer un mail.
 */
public class TestMail {
   private final static String MAILER_VERSION = "Java";
   public static boolean envoyerMailSMTP(String serveur, boolean debug) {
         boolean result = false;
         try {
                  Properties prop = System.getProperties();
                  prop.put("mail.smtp.host", serveur);
                  Session session = Session.getDefaultInstance(prop,null);
                  Message message = new MimeMessage(session);
                  message.setFrom(new InternetAddress("moi@chez-moi.fr"));
                  InternetAddress[] internetAddresses = new InternetAddress[1];
                  internetAddresses[0] = new InternetAddress("moi@chez-moifr");
                  message.setRecipients(Message.RecipientType.TO,internetAddresses);
                  message.setSubject("Test");
                  message.setText("test mail");
                  message.setHeader("X-Mailer", MAILER_VERSION);
                  message.setSentDate(new Date());
                  session.setDebug(debug);
                  Transport.send(message);
                  result = true;
         } catch (AddressException e) {
                  e.printStackTrace();
         } catch (MessagingException e) {
                  e.printStackTrace();
         }
         return result;
   }

   public static void main(String[] args) {
```

```
        TestMail.envoyerMailSMTP("10.10.50.8",true);
    }
}
```

javac −classpath activation.jar;mail.jar;smtp.jar %1.java

java −classpath .;activation.jar;mail.jar;smtp.jar %1

42.5. Récupérer les messages d'un serveur POP3

Cette section sera développée dans une version future de ce document

42.6. Les fichiers de configuration

Ces fichiers permettent d'enregistrer des implementations de protocoles supplémentaires et des valeurs par défaut. Il existe 4 fichiers répartis en deux catégories :

- javamail.providers et javamail.default.providers
- javamail.address.map et javamail.default.address.map

JavaMail recherche les informations contenues dans ces fichiers dans l'ordre suivant :

1. $JAVA_HOME/lib

2. META−INF/javamail.xxx dans le fichier jar de l'application

3. MATA−INF/javamail.default.xxx dans le fichier jar de javamail

Il est ainsi possible d'utiliser son propre fichier sans faire de modification dans le fichier jar de JavaMail. Cette utilisation peut se faire sur le poste client ou dans le fichier jar de l'application, ce qui offre une grande souplesse.

42.6.1. Les fichiers javamail.providers et javamail.default.providers

Ce sont deux fichiers au format texte qui contiennent la liste et la configuration des protocoles dont le système dispose d'une implémentation. L'application peut ainsi rechercher la liste des protocoles utilisables.

Chaque procotole est défini en utilisant des attributs avec la forme nom=valeur suivi d'un point virgule. Cinq attributs sont définis (leur nom doit être en minuscule) :

Nom de l'attribut	Rôle	Présence
protocol	nom du protocole	obligatoire
type	type protocole : « store » ou « transport »	obligatoire
class	nom de la classe contenant l'implémentation du protocole	obligatoire

vendor	nom du fournisseur	optionnelle
version	numero de version	optionnelle

Exemple : le contenu du fichier META−INF/javamail.default.providers

```
# JavaMail IMAP provider Sun Microsystems, Inc
protocol=imap; type=store; class=com.sun.mail.imap.IMAPStore; vendor=Sun Microsystems, Inc;
# JavaMail SMTP provider Sun Microsystems, Inc
protocol=smtp; type=transport; class=com.sun.mail.smtp.SMTPTransport; vendor=Sun Microsystems, Inc;
# JavaMail POP3 provider Sun Microsystems, Inc
protocol=pop3; type=store; class=com.sun.mail.pop3.POP3Store; vendor=Sun Microsystems, Inc;
```

42.6.2. Les fichiers javamail.address.map et javamail.default.address.map

Ce sont deux fichiers au format texte qui permettent d'associer un type de transport avec un protocole. Cette association se fait sous la forme nom=valeur suivi d'un point virgule.

Exemple : le contenu du fichier META−INF/javamail.default.address.map

```
rfc822=smtp
```

43. Les EJB (Entreprise Java Bean)

Les Entreprise Java Bean ou EJB sont des composants serveurs donc non visuels qui respectent les spécifications d'un modèle édité par Sun. Ces spécifications définissent une architecture, un environnement d'exécution et un ensemble d'API.

Le respect de ces spécifications permet d'utiliser les EJB de façon indépendante du serveur d'applications J2EE dans lequel ils s'exécutent, du moment ou le code de mise en oeuvre des EJB n'utilisent pas d'extensions proposées par un serveur d'applications particulier.

Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises.

Une des principales caractéristiques des EJB est de permettre aux développeurs de se concentrer sur les traitements orientés métiers car les EJB et l'environnement dans lequel ils s'exécutent prennent en charge un certain nombre de traitements tel que la gestion des transactions, la persistance des données, la sécurité, ...

Physiquement, un EJB est un ensemble d'au moins deux interfaces et une classe regroupées dans un module contenant un descripteur de déploiement particulier.

Pour obtenir des informations complémentaires sur les EJB, il est possible de consulter le site de Sun : java.sun.com/products/ejb

Il existe plusieurs versions des spécifications des E.J.B. :

- 1.0 :
- 1.1 :
- 2.0 :

Remarque : dans ce chapitre, le mot bean sera utilisé comme synonyme de EJB.

Ce chapitre contient plusieurs sections :

La suite de ce chapitre sera développée dans une version future de ce document

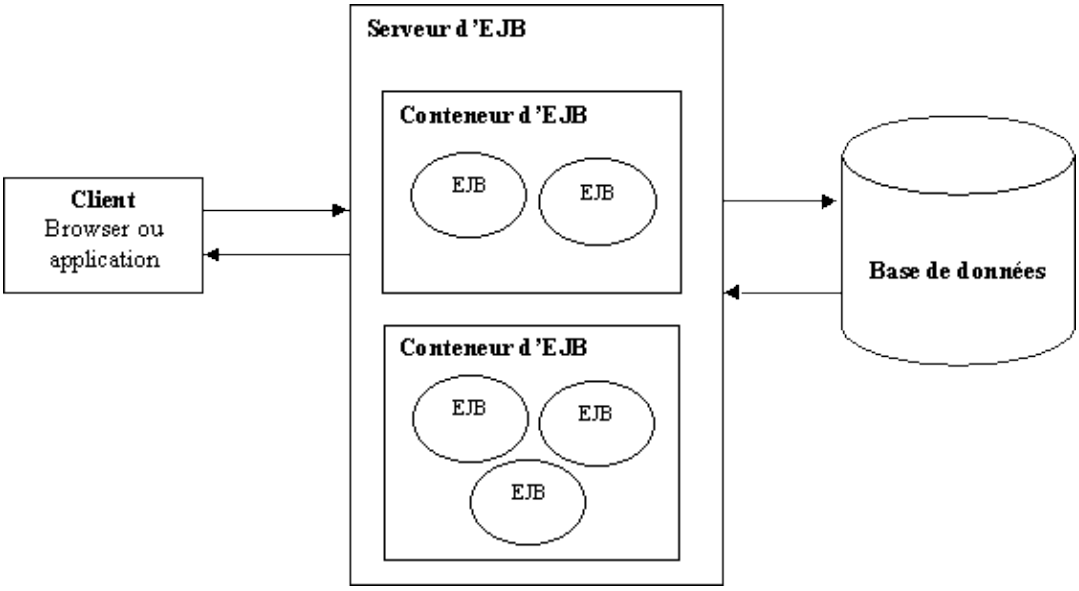
43.1. Présentation des EJB

Les EJB sont des composants et en tant que tel, ils possèdent certaines caractéristiques comme la réutilisabilité, la possibilité de s'assembler pour construire une application, etc ... Les EJB et les beans n'ont en commun que d'être des composants. Les java beans sont des composants qui peuvent être utilisés dans toutes les circonstances. Les EJB doivent obligatoirement s'exécuter dans un environnement serveur dédié.

Les EJB sont parfaitement adaptés pour être intégrés dans une architecture trois tiers ou plus. Dans une telle architecture, chaque tiers assure une fonction particulière :

- le client « léger » assure la saisie et l'affichage des données
- sur le serveur, les objets métiers contiennent les traitements. Les EJB sont spécialement conçus pour constituer de telles entités.
- une base de données assure la persistance des informations

Les EJB s'exécutent dans un environnement particulier : le serveur d'EJB. Celui ci fournit un ensemble de fonctionnalités utilisées par un ou plusieurs conteneurs d'EJB qui constituent le serveur d'EJB. En réalité, c'est dans un conteneur que s'exécute un EJB et il lui est impossible de s'exécuter en dehors.



Le conteneur d'EJB propose un certain nombre de services qui assurent la gestion :

- du cycle de vie du bean
- de l'accès au bean
- de la sécurité d'accès
- des accès concurrents
- des transactions

Les entités externes au serveur qui appellent un EJB ne communiquent pas directement avec celui ci. Les accès au EJB par un client se fait obligatoirement via le conteneur. Un objet héritant de EJBObject assure le dialogue entre ces entités et les EJB via le conteneur. L'avantage de passer par le conteneur est que celui ci peut utiliser les services qu'il propose et libérer ainsi le développeur de cette charge de travail. Ceci permet au développeur de se concentrer sur les traitements métiers proposés par le bean.

Il existe de nombreux serveurs d'EJB commerciaux : BEA Weblogic, IBM Webpshere, Sun IPlanet, Macromedia JRun, Borland AppServer, etc ... Il existe aussi des serveurs d'EJB open source dont les plus avancés sont JBoss et Jonas.

43.1.1. Les différents types d'EJB

Il existe deux types d'EJB : les beans de session (session beans) et les beans entité (les entity beans). Depuis la version 2.0 des EJB, il existe un troisième type de bean : les beans orienté message (message driven beans). Ces trois types de

bean possèdent des points communs notamment celui de devoir être déployés dans un conteneur d'EJB.

Les session beans peuvent être de deux types : sans état (stateless) ou avec état (statefull).

Les beans de session sans état peuvent être utilisés pour traiter les requêtes de plusieurs clients. Les beans de session avec état ne sont accessibles que lors d'un ou plusieurs échanges avec le même client. Ce type de bean peut conserver des données entre les échanges avec le client.

Les beans entité assurent la persistance des données. Il existe deux types d'entity bean :

- persistance gérée par le conteneur (CMP : Container Managed Persistence)
- persistance gérée par le bean (BMP : Bean Managed Persistence).

Avec un bean entité CMP (container−managed persistence), c'est le conteneur d'EJB qui assure la persistance des données. Un bean entité BMP (bean−managed persistence), assure lui même la persistance des données grâce à du code inclus dans le bean.

La spécification 2.0 des EJB définit un troisième type d'EJB : les beans orientés message (message−driven beans).

43.1.2. Le développement d'un EJB

Le cycle de développement d'un EJB comprend :

- la création des interfaces et des classes du bean
- le packaging du bean sous forme de fichier archive jar
- le déploiement du bean dans un serveur d'EJB
- le test du bean

La création d'un bean nécessite la création d'au minimum deux interfaces et une classe pour respecter les spécifications de Sun : la classe du bean, l'interface remote et l'interface home.

L'interface remote permet de définir l'ensemble des services fournis par le bean. Cette interface étend l'interface EJBObject. Dans la version 2.0 des EJB, l'API propose une interface supplémentaire, EJBLocalObject, pour définir les services fournis par le bean qui peuvent être appellés en local par d'autres beans. Ceci permet d'éviter de mettre en oeuvre toute une mécanique longue et couteuse en ressources pour appeler des beans s'exécutant dans le même conteneur.

L'interface home permet de définir l'ensemble des services qui vont permettre la gestion du cycle de vie du bean. Cette interface étend l'interface EJBHome.

La classe du bean contient l'implémentation des traitements du bean. Cette classe implémente les méthodes déclarées dans les interfaces home et remote. Les méthodes définissant celle de l'interface home sont obligatoirement préfixées par "ejb".

L'accès aux fonctionnalités du bean se fait obligatoirement par les méthodes définies dans les interfaces home et remote.

Il existe un certain nombre d'API qu'il n'est pas possible d'utiliser dans un EJB :

- les threads
- flux pour des entrées/sorties
- du code natif
- AWT et Swing

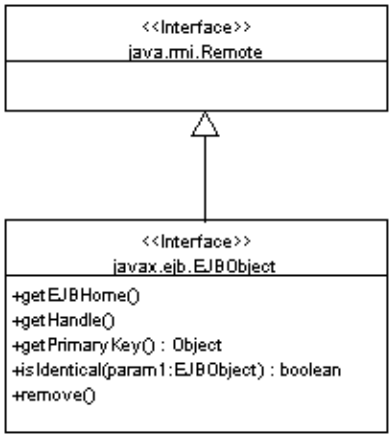
43.1.3. L'interface remote

L'interface remote permet de définir les méthodes qui contiendront les traitements proposés par le bean. Cette interface doit étendre l'interface javax.ejb.EJBObject.

Exemple :

```
package com.moi.ejb;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;


public interface MonPremierEJB extends EJBObject {
  public String message() throws RemoteException;
}
```

Toutes les méthodes définies dans cette interface doivent obligatoirement respecter les spécifications de RMI et déclarer qu'elles peuvent lever une exception de type RemoteException.



L'interface javax.ejb.EJBObject définit plusieurs méthodes qui seront donc présentes dans tous les EJB :

- EJBHome getEJBHome() throws java.rmi.RemoteException : renvoie une référence sur l'objet Home
- Handle getHandle() throws java.rmi.RemoteException : renvoie un objet permettant de sérialiser le bean
- Object getPrimaryKey() throws java.rmi.RemoteException : renvoie une référence sur l'objet qui encapsule la clé primaire d'un bean entité
- boolean isIdentical(EJBObject) throws java.rmi.RemoteException : renvoie un boolean qui précise si le bean est identique à l'instance du bean fourni en paramètre. Pour un bean session sans état, cette méthode renvoie toujours true. Pour un bean entité, la méthode renvoie true si la clé primaire des deux beans est identique.
- void remove() throws java.rmi.RemoteException, javax.ejb.RemoveException : cette méthode demande la destruction du bean. Pour un bean entité, elle provoque la suppression des données coorespondantes dans la base de données.

43.1.4. L'interface home

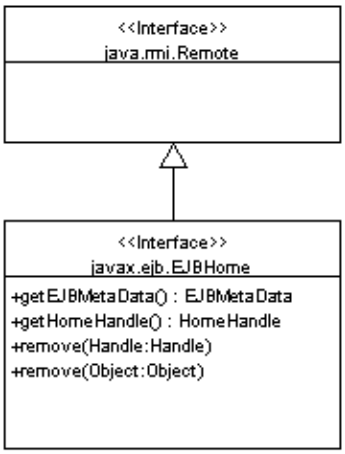
L'interface home permet de définir des méthodes qui vont gérer le cycle de vie du bean. Cette interface doit étendre l'interface EJBHome.

La création d'une instance d'un bean se fait grâce à une ou plusieurs surchages de la méthode create(). Chacune de ces méthodes renvoie une instance d'un objet du type de l'interface remote.

Exemple :

```
package com.moi.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
```

```
import javax.ejb.EJBHome;


public interface MonPremierEJBHome extends EJBHome {
  public MonPremierEJB create() throws CreateException, RemoteException;
}
```



L'interface javax.ejb.EJBHome définit plusieurs méthodes :

- EJBMetaData getEJBMetaData() throws java.rmi.RemoteException
- HomeHandle getHomeHandle() throws java.rmi.RemoteException : renvoie un objet qui permet de sérialiser l'objet implémentant l'interface EJBHome
- void remove(Handle) throws java.rmi.RemoteException, javax.ejb.RemoveException : supprime le bean
- void remove(Object) throws java.rmi.RemoteException, javax.ejb.RemoveException : supprime le bean entité dont l'objet encapsulant la clé primaire est fourni en paramètre

La ou les méthodes à définir dans l'interface home dépendent du type d'EJB:

Type de bean	Méthodes à définir
bean session sans état	une seule méthode create() sans paramètre
bean session avec état	une ou plusieurs méthodes create()
bean entité	aucune ou plusieurs méthodes create() et une ou plusieurs méthodes finder()

43.2. Les EJB session

Un EJB session est un EJB de service dont la durée de vie correspond à un échange avec un client. Ils contiennent les règles métiers de l'application.

Il existe deux types d'EJB session : sans état (stateless) et avec état (statefull).

Les EJB session statefull sont capables de conserver l'état du bean dans des variables d'instance durant toute la conversation avec un client. Mais ces données ne sont pas persistantes : à la fin de l'échange avec le client, l'instance de l'EJB est détruite et les données sont perdues.

Les EJB session stateless ne peuvent pas conserver de telles données entre chaque appel du client.

Il ne faut pas faire appel directement aux méthodes create() et remove() de l'EJB. C'est le conteneur d'EJB qui se charge de la gestion du cycle de vie de l'EJB et qui appelle ces méthodes. Le client décide simplement du moment de la création et de la suppression du bean en passant par le conteneur.

Une classe qui encapsule un EJB session doit implémenter l'interface javax.ejb.SessionBean. Elle ne doit pas implémenter les interfaces home et remote mais elle doit définir les méthodes déclarées dans ces deux interfaces.

La classe qui implémente le bean doit définir les méthodes définies dans l'interface remote. La classe doit aussi définir la méthode ejbCreate(), ejbRemove(), ejbActivate(), ejbPassivate et setSessionContext().

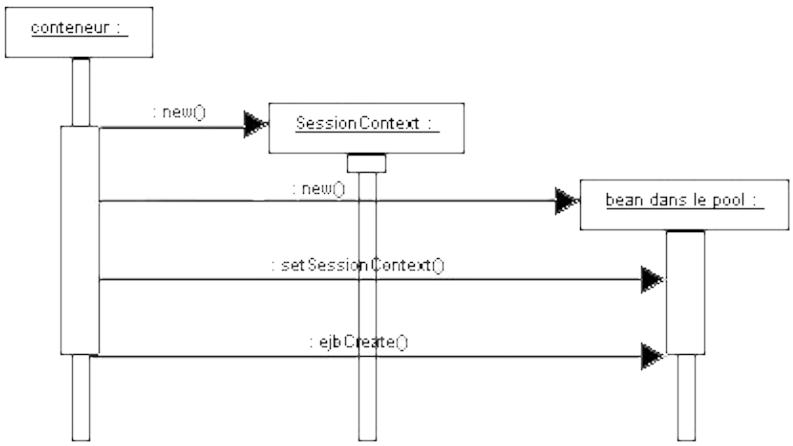
Le méthode ejbRemove() est appellée par le conteneur lors de la suppression de l'instance du bean.

Pour permettre au serveur d'application d'assurer la monter en charge des différentes applications qui s'exécutent dans ces conteneurs, celui ci peut momentanement liberer de la mémoire en déchargeant un ou plusieurs beans. Cette action consiste à sérialiser le bean sur le système de fichiers et de le déssérialiser pour sa remonter en mémoire. Lors de ces deux actions, le conteneur appel respectivement les méthodes ejbPassivate() et ejbActivate().

43.2.1. Les EJB session sans état

Ce type de bean propose des services sous la forme de méthodes. Il ne peut pas conserver de données entre deux appels de méthodes. Les données provenant du client nécessaires aux traitements d'une méthode doivent obligatoirement être fournies en paramètre de la méthode.

Les services proposés par ces beans peuvent être gérés dans un pool par le conteneur pour améliorer les performances puisqu'ils sont indépendants du client qui les utilisent. Le pool contient un certain nombre d'instances du bean. Toutes ces instances étant "identiques", il suffit au conteneur d'ajouter ou de supprimer de nouvelles instances dans le pool selon les variations de la charge du serveur d'application. Il est donc inutile au serveur de sérialiser un EJB session sans état. Il suffit simplement de déclarer les méthodes ejbActivate() et ejbPassivate() sans traitements.



Le conteneur s'assure qu'un même bean ne recevra pas d'appel de méthode de la part de deux clients différents en même temps.

Exemple :

```
package com.moi.ejb;

import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;


public class MonPremierEJBBean implements SessionBean {

  public String message() {
    return "Bonjour";
  }

  public void ejbActivate() {
  }

  public void ejbPassivate() {
```

```
  }

  public void ejbRemove() {
  }

  public void setSessionContext(SessionContext arg0) throws EJBException, RemoteException {
  }

  public void ejbCreate() {
  }
}
```

43.2.2. Les EJB session avec état

Ce type de bean fourni aussi un ensemble de traitements via ces méthodes mais il a la possibilité de conserver des données entre les différents appels de méthodes d'un même client. Une instance particulière est donc dédiée à chaque client qui sollicite ces services et ce tout au long du dialogue entre les deux entités.

Les données conservées par le bean sont stockées dans les variables d'instances du bean. Les données sont donc conservées en mémoire. Généralement, les méthodes proposées par le bean permettent de consulter et mettre à jour ces données.

Dans un EJB session avec état il est possible de définir plusieurs méthodes permettant la création d'un tel EJB. Ces méthodes doivent obligatoirement commencer par ejbCreate.

Les méthodes ejbPassivate() et ejbActivate() doivent définir et contenir les éventuels traitements lors de leur appel par le conteneur. Celui ci appel ces deux méthodes respectivement lors de la sérialisation du bean et sa dessérialisation. La méthode ejbActivate() doit contenir les traitements nécessaires à la restitution du bean dans un état utilisable après la dessérialisation.

Le cycle de vie d'un ejb avec état est donc identique à celui d'un bean sans état avec un état supplémentaire lorsque celui est sérialisé. La fin du bean peut être demandée par le client lorsque celui ci utilise la méthode remove(). Le conteneur invoque la méthode ejbRemove() du bean avant de supprimer sa référence.

Certaines méthodes métiers doivent permettre de modifier les données stockées dans le bean.

43.3. Les EJB entité

Ces EJB permettent de représenter et de gérer des données enregistrées dans une base de données. Ils implémentent l'interface EntityBean.

L'avantage d'utiliser un tel type d'EJB plutot que d'utiliser JDBC ou de développer sa propre solution pour mapper les données est que certains services sont pris en charge par le conteneur.

Les beans entité assurent la persistance des données en représentant tout au partie d'une table ou d'une vue. Il existe deux types de bean entité :

- persistance gérée par le conteneur (CMP : Container Managed Persistence)
- persistance gérée par le bean (BMP : Bean Managed Persistence).

Avec un bean entité CMP (container−managed persistence), c'est le conteneur d'EJB qui assure la persistance des données grâce aux paramètres fournis dans le descripteur de déploiement du bean. Il se charge de toute la logique des traitements de synchronisation entre les données du bean et les données dans la base de données.

Un bean entité BMP (bean−managed persistence), assure lui même la persistance des données grâce à du code inclus dans les méthodes du bean.

Plusieurs clients peuvent accéder simultanément à un même EJB entity. La gestion des transactions et des accès concurrents est assurée par le conteneur.

43.4. Les outils pour développer et mettre oeuvre des EJB

43.4.1. Les outils de développement

Plusieurs EDI (Environnement de Développement Intégré) commerciaux fournissent dans leur version Entreprise des outils pour développer et tester des EJB. On peut citer Jbuilder d'Inprise/Borland ou Visual Age Java et WSAD d'IBM. Mais ces produits sont très coûteux pour une utilisation personnelle.

Il existe quelques solutions libres utilisables : même si elles sont moins abouties que les outils commerciaux elles proposent cependant des fonctionnalités particulièrement intéressantes.

43.4.2. Les serveurs d'EJB

43.4.2.1. Jboss

JBoss est un serveur d'EJB Open Source écrit en java.

Il peut être téléchargé sur www.jboss.org.

JBoss nécessite la présence du J.D.K. 1.3.

Pour l'installer, il suffit de dézipper l'archive et de copier son contenu dans un répertoire , par exemple : c\jboss

Pour lancer le serveur, il suffit d'exécuter la commande :

java −jar run.jar

Les EJB à déployer doivent être mis dans le répertoire deploy. Si le répertoire existe au lancement du serveur, les EJB seront automatiquement déployés dès qu'ils seront insérés dans ce répertoire.

43.5. Le déploiement des EJB

Pour permettre le déploiement d'un EJB, il faut définir un fichier DD (deployement descriptor) qui contient des informations sur le bean. Ce fichier au format XML permet de donner au conteneur d'EJB des caractéristiques du bean.

Un EJB doit être déployé sous forme d'une archive jar qui doit contenir un fichier qui est le descripteur de déploiement et toutes les classes qui composent chaque EJB (interfaces home et remote, les classes qui implémentent ces interfaces et toutes les autres classes nécessaires aux EJB).

Une archive ne doit contenir qu'un seul descripteur de deploiement pour tous les EJB de l'archive. Ce fichier au format XML doit obligatoirement être nommé ejb−jar.xml.

L'archive doit contenir un répertoire META−INF (attention au respect de la casse) qui contiendra lui même le descripteur de déploiement.

Le reste de l'archive doit contenir les fichiers .class avec toute l'arborescence des répertoires des packages.

43.5.1. Le descripteur de déploiement

Le descripteur de déploiement est un fichier au format XML qui permet de fournir au conteneur des informations sur les beans à déployer. Le contenu de ce fichier dépend du type de beans à déployer.

43.5.2. Le mise en package des beans

Une fois toutes les classes et le fichier de déploiement écrit, il faut les rassembler dans une archive .jar afin de pouvoir les déployer dans le conteneur.

43.6. L'appel d'un EJB par un client

Un client peut être une entité de toute forme : application avec ou sans interface graphique, un bean, une servlet ou une JSP ou un autre EJB.

Un EJB étant un objet distribué, son appel utilise RMI.

Le stub est une représentation locale de l'objet distant. Il implémente l'interface remote mais contient une connection réseau pour accéder au skeleton à l'objet distant.

Le mode d'appel d'un EJB suit toujours la même logique :

- obtenir une référence qui implémente l'interface home de l'EJB grace à JNDI
- créer une instance qui implémente l'interface remote en utilisant la référence précédemment acquise
- appelle de la ou des méthodes de l'EJB
-

43.6.1. Exemple d'appel d'un EJB session

L'appel d'un EJB session avec ou sans état suit la même logique.

Il faut tout d'abord utiliser un objet du type InitialContext pour pouvoir interroger JNDI. Cet objet nécessite qu'on lui fournisse des informations dont le nom de la classe à utiliser comme fabrique et l'url du serveur JNDI.

Cet objet permet d'obtenir une reférence sur le bean enregistré dans JNDI. A partir de cette référence, il est possible de créer un objet qui implémente l'interface home. Un appel à la méthode create() sur cet objet permet de créer un objet du type de l'EJB. L'appel des méthodes de cet objet entraine l'appel des méthodes de l'objet EJB qui s'exécute dans le conteur.

Exemple :

```
package testEJBClient;

import java.util.*;
import javax.naming.*;


public class EJBClient {

  public static void main(String[] args) {
    Properties ppt = null;
    Context ctx = null;
```

```
    Object ref = null;
    MonPremierBeanHome home = null;
    MonPremierBean bean = null;

    try {
      ppt = new Properties();
      ppt.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
      ppt.put(Context.PROVIDER_URL, "localhost:1099");
      ctx = new InitialContext(ppt);
      ref = ctx.lookup("MonPremierBean");
      home = (MonPremierBeanHome) javax.rmi.PortableRemoteObject.narrow(ref,
        MonPremierBeanHome.class);
      bean = home.create();
      System.out.println("message = " + bean.message());
      bean.remove();
    }
    catch (Exception e) {
       e.printStackTrace();
    }
  }

}
```

43.7. Les EJB orientés messages

Ces EJB sont différents des deux types d'EJB car ils répondent à des invocations de façon asynchrone. Ils permettent de réagir à l'arrivée de messages fournis par un M.O.M. (middleware oriented messages).

44. Les services web

La suite de ce chapitre sera développée dans une version future de ce document

Les services web permettent l'appel d'une méthode d'un objet distant en utilisant un protocol web pour transport (http en général) et XML pour formatter les échanges. Les services web fonctionnent sur le principe du client serveur :

- un client appelle les service web
- le serveur traite la demande et renvoie le résultat au client
- le client utilise le résultat

L'appel de méthodes distantes n'est pas une nouveauté mais la grande force des services web est d'utiliser des standards ouverts et reconnus : HTTP et XML. L'utilisation de ces standards permet d'écrire des services web dans plusieurs langages et de les utiliser sur des systèmes d'exploitation différents.

Les services web utilisent des échanges de messages au format XML.

Il existe deux types de services web :

- synchrone : appel de méthodes (SOAP)
- asynchrone : échange de messages (SOAP, ebXML)

Les services web ne sont pas encore complètement matures à cause de la jeunesse des technologies utilisées pour les mettre en oeuvre. Il reste encore de nombreux domaines à enrichir (sécurité, gestion des transactions, workflow, ...). Des technologies pour répondre à ces besoins sont en cours de développement.

Initialement, Sun a proposé un ensemble d'outils et d'API pour permettre le développement de services web avec Java. Cet ensemble se nomme JWSDP (Java Web Services Developer Pack) et il existe deux versions 1.1 et 1.2. Depuis Sun à intégré la plupart de ces API permettant le développement de services web dans les spécifications de J2EE version 1.4.

Ce chapitre contient plusieurs sections :

44.1. Les technologies utilisées

Les services web utilisent trois technologies :

- SOAP (Simple Object Access Protocol) pour le service d'invocation : il permet l'échange de messages dans un format particulier
- WSDL (Web Services Description Language) pour le service de description : il permet de décrire les services web
- UDDI (Universal Description Discovery and Integration) pour le service de publication : il permet de référencer les services web

44.1.1. SOAP

SOAP est une norme de communication qui standardise l'échange de messages en utilisant un protocole de communication et XML pour formatter les données. Le protocole le plus utilisé est HTTP pour sa facilité de mise en oeuvre mais d'autres protocoles peuvent être utilisés tel que FTP ou SMTP. En fait, tous les protocoles capables de véhiculer un flux d'octets peuvent être utilisés.

SOAP se veut simple à utiliser et extensible.

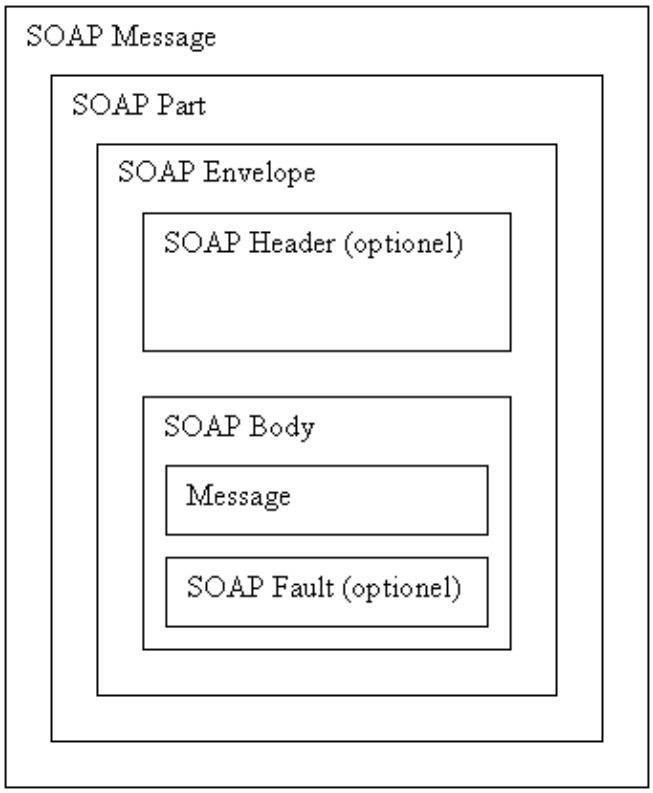
La spécification courante de SOAP est la 1.1. Le w3c travaille sur la version 1.2.

SOAP peut être utilisé :

- pour l'appel de méthodes (SOAP RPC)
- pour l'échange de message (SOAP Messaging)

SOAP définit la structure principale du message, dite « enveloppe » qui contient deux parties :

- en tête (Header) : facultatif
- le corps (Body) : obligatoire



Le corps est composé d'un ou plusieurs blocs. Un bloc contient des données ou un appel de méthode avec ces paramètres.

Tous ces éléments sont codés dans le message XML avec un tag particulier mettant en oeuvre un espace de nommage particulier défini dans les spécifications de SOAP.

Un message SOAP peut aussi contenir des pièces jointes contenues chacunes dans une partie optionnelle nommée AttachmentPart. Ces parties sont au même niveau que la partie SOAP Part.

SOAP définit aussi l'encodage pour les différents types de données qui est basé sur la technologie schéma XML du W3C. Les données peuvent être de type simple (chaine, entier, flottant, ...) ou de type composé.

Les types simples peuvent être

- un type de base : string, int, float, ...
- une énumeration
- un tableau d'octets (array of byte)

Les types composés

- une structure (Struct)
- un tableau (Array)

La partie SOAP Fault permet d'indiquer qu'une erreur est survenue lors des traitements du service web. Cette partie peut être composée de 4 éléments :

- faultcode : indique le type de l'erreur (VersionMismatch en cas d'incompatibilité avec la version de SOAP utilisée, MustUnderstand en cas de problème dans le header du message, Client en cas de manque d'informations de la part du client, Server en cas de problème d'exécution des traitements par le serveur)
- faultstring : message décrivant l'erreur
- faultactor : URI de l'élement ayant déclenché l'erreur
- faultdetail

Pour l'appel de méthodes, plusieurs informations sont nécessaires :

- l'URI de l'objet à utiliser
- le nom de la méthode
- éventuellement le ou les paramètres

44.1.2. WSDL

WSDL est une norme qui utilise XML pour décrire des services web.

L'utilisation de WSDL n'est pas obligatoire mais elle est utilisée par la plupart des outils pour faciliter la génération automatique de certains objets dont le but est de faciliter l'utilisation du service.

Il n'y a pas d'API pour manipuler directement un fichier WSDL : un spécification le permettant est en cours de développement par le JCP sous la JSR 110 nommée JWSDL.

44.1.3. UDDI

UDDI (Universal Description, Discovery and Integration) est un protocole et un ensemble de services pour utiliser un annuaire afin de stocker les informations concernant les services web et de permettre à un client de les retrouver.

44.2. Les API Java liées à XML pour les services web

L'API de base pour le traitement de document XML avec java est JAXP. JAXP regroupe un ensemble d'API pour traiter des documents XML avec SAX et DOM et les modifier avec XSLT. Cette API est indépendante de tout parseur. JAXP est détaillé dans le chapitre Java et XML.

D'autres API sont spécifiques au développement de service web :

- JAX−RPC (JSR−101) : permet l'appel de procédures distantes en utilisant SOAP (Remote Procedure Call)
- JAXM (JSR−67) : permet l'envoie de messages (en utilisant SAAJ)
- JAXR : permet l'accès au service de registre de façon standard (UDDI)
- SAAJ (SOAP with Attachment API for Java) : permet l'envoi et la réception de messages respectant les normes SOAP et SOAP with Attachment

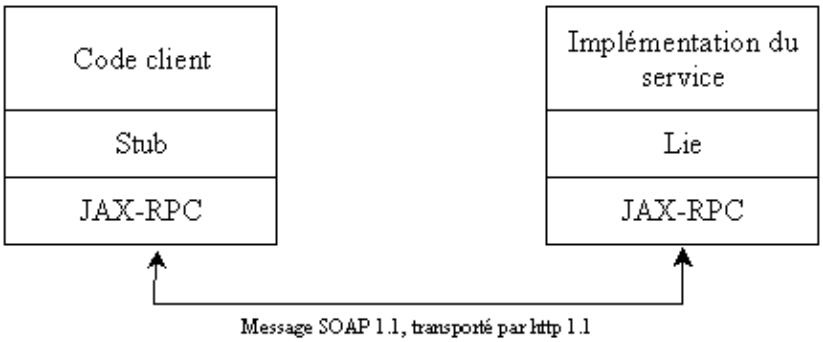
44.2.1. JAX−RPC

JAX−RPC est l'acronyme de Java API for XML based RPC. Cette API permet facilement l'appel de méthodes distantes et la réception de leur réponse en utilisant SOAP 1.1 et HTTP 1.1. Cette facilité permet de s'affranchir d'une mise en oeuvre détaillée de SOAP pour réaliser les opérations.

Cette API a été développée par le JCP sous la JSR 101.

Le grand avantage de cette API est de masquer un grand nombre de détails de l'utilisation de SOAP notamment en ce qui concerne le codage en XML du message et ainsi de rendre cette API facile à utiliser.

L'utilisation de JAX−RPC est similaire à celle de RMI : le code du client appel les méthodes à partir d'un objet local nommé stub. Cet objet se charge de dialoguer avec le serveur et de coder et décoder les messages SOAP échangés.



Côté serveur, un objet similaire nommé lie permet de réaliser le même type d'opération côté serveur.

La principale différence entre RMI et les services web est que RMI ne peut être utilisé qu'avec Java alors que les services web sont interopérables grâce à XML. Ainsi un client écrit en Java peut utiliser un services web développé avec .Net et vice et versa.

L'utilisation de JAX−RPC se fait en plusieurs étapes :

1. Définition de l'interface du service (écrite manuellement ou générée automatiquement par un outil à partir de la description du service (WSDL)).

Exemple :

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
public interface MonWS extends Remote {
  public String getMessage(String nom) throws RemoteException;
}
```

Cette interface doit étendre l'interface java.rmi.Remote.

Toutes les méthodes définies dans l'interface doivent au minimum déclarer la possibilité de lever une exception de type java.rmi.RemoteException. Chaque méthode peut aussi déclarer d'autres exceptions dans sa définition du moment que ces exceptions héritent de la classe java.lang.Exception.

Les méthodes peuvent sans restriction utiliser des types primitifs et l'objet String pour les paramètres et la valeur de retour. Pour les autres types, il existe dans les spécifications une liste minimale prédéfinie de ceux utilisables.

Une implémentation particulière peut cependant proposer le support d'autres types. Par exemple, l'implémentantion de référence propose le support de la plupart des classes de l'API Collection : ArrayList, HashMap, HashTable, LinkedList, TreeMap, TreeSet, Vector, ... Attention cependant dans ce cas, à la perte de la portabilité lors de l'utilisation d'une autre implémentation.

2. Ecriture de la classe d'implémentation du service

C'est une simple classe Java qui implémente l'interface définie précédemment.

Exemple :

```
public class MonWS_Impl implements MonWS {
  public String getMessage(String nom) {
    return new String("Bonjour " + nom);
  }
}
```

Cette implémentation doit obligatoirement implémenter l'interface définie précédemment et posséder un constructeur sans paramètre : dans l'exemple, celui ci sera généré lors de la compilation car il n'y a pas d'autre constructeur défini.

Il est inutile dans l'implementation des méthodes de déclarées la levée de l'exception de type RemoteException. C'est lors de l'invocation de la méthode par JAX−RPC que cette exception pourra être levée.

3. Déploiement du service

L'API JAX−RPC est regroupée dans plusieurs sous packages du package javax.xml.rpc

L'invocation de méthodes côté client se faire de manière synchrone avec JAX−RPC : le client fait appel au service et se met en attente jusqu'à la reception de la réponse

44.2.2. JAXM

L'API JAXM (Java API for XML Messaging) propose de standardiser l'échange de messages. JAXM a été developpé sous la JSR−067.

Les classes de cet API sont regroupées dans le package javax.xml.messaging.

44.2.3. SAAJ

L'API SAAJ (SOAP with Attachment API for Java) permet l'envoi et la réception de messages respectant les normes SOAP 1.1 et SOAP with attachments : cette API propose un niveau d'abstraction assez élévé permettant de simplifier l'usage de SOAP.

Les classes de cette API sont regroupées dans le package javax.xml.soap.

Originellement, cette API était incluse dans JAXM. Depuis la version 1.1, elles ont été séparées.

SAAJ propose des classes qui encapsulent les différents éléments d'un message SOAP : SOAPMessage, SOAPPart, SOAPEnvelope, SOAPHeader et SOAPBody.

Tous les échanges de messages avec SOAP utilisent une connexion encapsulée dans la classe SOAPConnection. Cette classe permet la connexion directe entre l'émetteur et le receveur du ou des messages.

44.2.4. JAXR

L'API JAXR (Java API for XML Registries) propose de standardiser l'utilisation de registres dans lesquels sont recensés les services web. JAXR permet notamment un accès aux registres de type UDDI.

44.3. Mise en oeuvre avec JWSDP

Le Java Web Services Developer Pack (JWSDP) est un ensemble d'outils et d'API qui permet de faciliter le développement des services web et des applications web avec Java. Il est possible de le télécharger sur le site de Sun : http://java.sun.com/webservices/.

Pour pouvoir l'utiliser il faut au minimum un jdk 1.3.1.

Le JWSDP contient les API particulières suivantes :

- Java XML Pack : Java API for XML Processing (JAXP), Java API for XML−based RPC (JAX−RPC), Java API for XML Messaging (JAXM), Java API for XML Registries (JAXR)
- Java Architecture for XML Binbing (JAXB)
- JavaServer Pages Standard Tag Library (JSTL)
- Java Secure Socket (JSSE)
- SOAP with Attachments API for Java (SAAJ)

Le JWSDP contient les outils suivants :

- Apache Tomcat
- Java WSDP Registry Server (serveur UDDI)
- Web application development tool
- Apache Ant

La plupart de ces éléments peuvent être installés manuellement séparement. Le JWSDP propose un pack qui les regroupent en une seule installation et propose en plus des outils spécifiquement dédiés au développement de services web.

44.3.1. Installation du JWSDP 1.1

Il faut télécharger sur le site de Sun le fichier jwsdp−1_1−windows−i586.exe et l'exécuter.

Un assistant guide l'installation :

- Cliquer sur "Suivant".
- Lire le contrat de licence, sélectionner "Approve" et cliquer sur "Suivant".
- Sélectionner le JDK à utiliser et cliquer sur "Suivant"
- Dans le cas de l'utilisation d'un proxy, il faut renseigner les informations le concernant. Cliquer sur "Suivant".
- Sélectionner le répertoire d'installation et cliquer sur "Suivant".
- Sélectionner le type d'installation et cliquer sur "Suivant".
- Il faut saisir un nom d'utilisation qui sera l'administrateur et son mot de passe et cliquer sur "Suivant".
- L'assitant affiche un récapitulatif des options choisies. Cliquer sur "Suivant".
- Cliquer sur "Suivant".
- Cliquer sur "Suivant".
- Cliquer sur "Fin".

44.3.2. Exécution

L'installation a créé une entrée dans le menu "Démarrer/Programmes".

Lancer le serveur d'application tomcat en utilisant l'option Start tomcat

Attention, les ports 8080 et 8081 ne doivent pas être occupés par un autre serveur.

Lancer un browser sur l'url http://localhost:8081/admin

Si la page ne s'affiche pas, il faut aller voir dans le fichier catalina.out contenu dans le répertoire logs ou a été installé le JWSDP.

Il faut saisir le nom de l'utilisateur et le mot de passe saisis lors de l'installation de JWSDP.

Cette console permet de modifier les paramètres du JWSDP.

44.3.3. Exécution d'un des exemples

Il faut créer un fichier build.properties dans le repertoire home (c:\document and settings\user_name) qui contient

username=

password=

Il faut s'assurer que le chemin C:\java\jwsdp−1_0_01\bin est en premier dans le classpath surtout si une autre version de Ant est déjà installée sur la machine

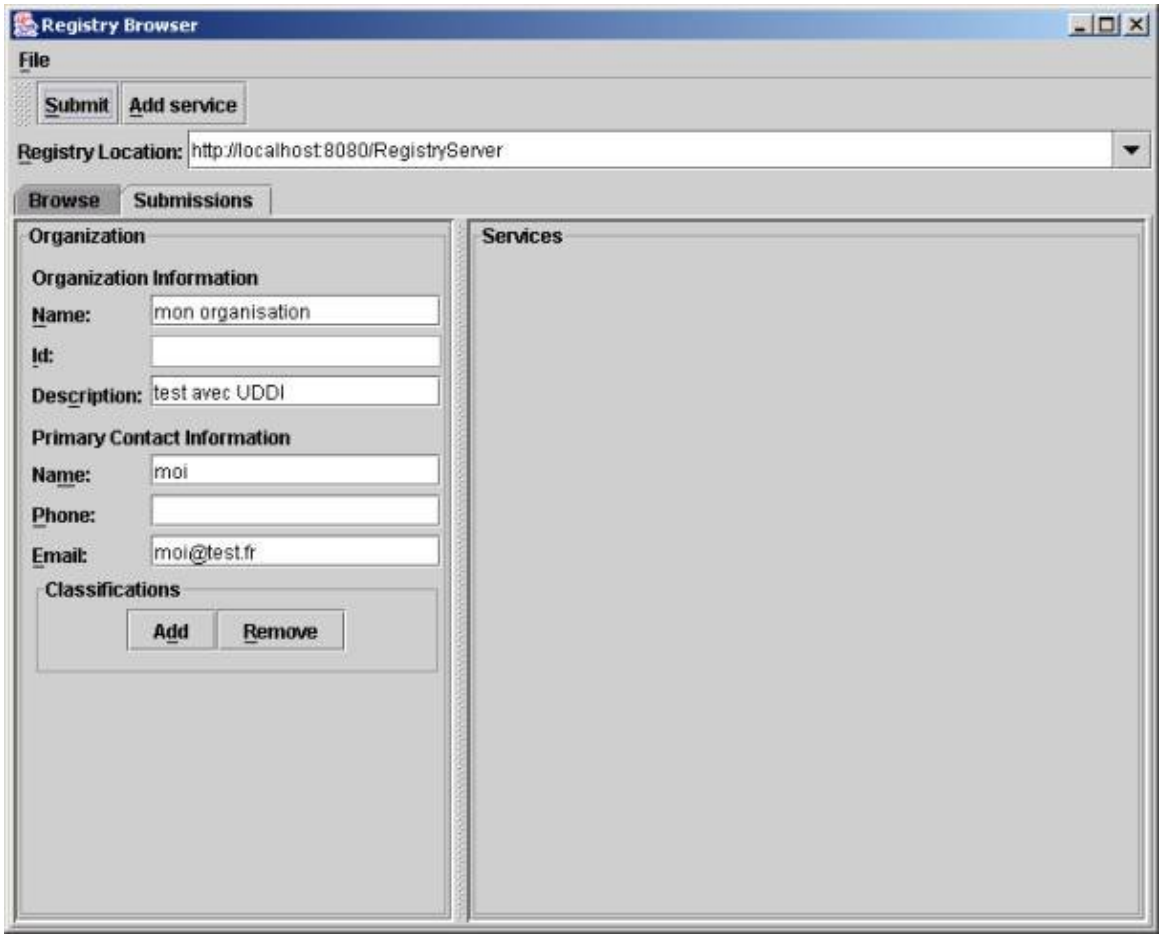
Il faut lancer Tomcat puis suivre les étapes proposées ci dessous :

Exemple :

```
C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hello>dir
Le volume dans le lecteur C s'appelle SYSTEM
Le numéro de série du volume est 18AE−3A71
Répertoire de C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hello
03/01/2003  13:37       <DIR>          .
03/01/2003  13:37       <DIR>          ..
01/08/2002  14:16                   309 build.properties
01/08/2002  14:17                   496 build.xml
01/08/2002  14:17                   222 config.xml
01/08/2002  14:16                 2 342 HelloClient.java
01/08/2002  14:17                 1 999 HelloIF.java
01/08/2002  14:16                 1 995 HelloImpl.java
01/08/2002  14:17                   545 jaxrpc−ri.xml
01/08/2002  14:17                   421 web.xml
               8 fichier(s)            8 329 octets
               2 Rép(s)     490 983 424 octets libres
C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hello>ant compile−server
Buildfile: build.xml
prepare:
     [echo] Creating the required directories....
    [mkdir] Created dir: C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hell
o\build\client\hello
    [mkdir] Created dir: C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hell
o\build\server\hello
    [mkdir] Created dir: C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hell
o\build\shared\hello
    [mkdir] Created dir: C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hell
o\build\wsdeploy−generated
    [mkdir] Created dir: C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hell
o\dist
    [mkdir] Created dir: C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hell
o\build\WEB−INF\classes\hello
compile−server:
     [echo] Compiling the server−side source code....
    [javac] Compiling 2 source files to C:\java\jwsdp−1_0_01\docs\tutorial\examp
les\jaxrpc\hello\build\shared
BUILD SUCCESSFUL
Total time: 7 seconds
C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hello>ant setup−web−inf
Buildfile: build.xml
setup−web−inf:
     [echo] Setting up build/WEB−INF....
   [delete] Deleting directory C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrp
c\hello\build\WEB−INF
     [copy] Copying 2 files to C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrp
c\hello\build\WEB−INF\classes\hello
     [copy] Copying 1 file to C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc
\hello\build\WEB−INF
     [copy] Copying 1 file to C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc
\hello\build\WEB−INF
BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\jwsdp−1_0_01\docs\tutorial\examples\jaxrpc\hello>ant package
Buildfile: build.xml
package:
     [echo] Packaging the WAR....
```

```
      [jar] Building jar: C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hel
lo\dist\hello-portable.war
BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>ant process-war
Buildfile: build.xml
set-ws-scripts:
process-war:
     [echo] Running wsdeploy....
     [exec] info: created temporary directory: C:\java\jwsdp-1_0_01\docs\tutoria
l\examples\jaxrpc\hello\build\wsdeploy-generated\jaxrpc-deploy-b5e49c
     [exec] info: processing endpoint: MyHello
     [exec] Note: sun.tools.javac.Main has been deprecated.
     [exec] 1 warning
     [exec] info: created output war file: C:\java\jwsdp-1_0_01\docs\tutorial\ex
amples\jaxrpc\hello\dist\hello-jaxrpc.war
     [exec] info: removed temporary directory: C:\java\jwsdp-1_0_01\docs\tutoria
l\examples\jaxrpc\hello\build\wsdeploy-generated\jaxrpc-deploy-b5e49c
BUILD SUCCESSFUL
Total time: 15 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>ant deploy
Buildfile: build.xml
deploy:
   [deploy] OK - Installed application at context path /hello-jaxrpc
   [deploy]
BUILD SUCCESSFUL
Total time: 7 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>ant generate-stubs
Buildfile: build.xml
set-ws-scripts:
prepare:
     [echo] Creating the required directories....
generate-stubs:
     [echo] Running wscompile....
     [exec] Note: sun.tools.javac.Main has been deprecated.
     [exec] 1 warning
BUILD SUCCESSFUL
Total time: 14 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>ant compile-client
Buildfile: build.xml
prepare:
     [echo] Creating the required directories....
compile-client:
     [echo] Compiling the client source code....
    [javac] Compiling 1 source file to C:\java\jwsdp-1_0_01\docs\tutorial\exampl
es\jaxrpc\hello\build\client
BUILD SUCCESSFUL
Total time: 4 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>ant jar-client
Buildfile: build.xml
jar-client:
     [echo] Building the client JAR  file....
      [jar] Building jar: C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hel
lo\dist\hello-client.jar
BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>ant run
Buildfile: build.xml
run:
     [echo] Running the hello.HelloClient program....
     [java] Hello Duke!
BUILD SUCCESSFUL
Total time: 5 seconds
C:\java\jwsdp-1_0_01\docs\tutorial\examples\jaxrpc\hello>
```

44.3.4. L'utilisation du JWSDP Registry Server



44.4. Mise en oeuvre avec Axis

Axis (Apache eXtensible Interaction System) est un projet open−source du groupe Apache. Son but est de proposer un ensemble d'outils pour faciliter le développement, le déploiement et l'utilisation des services web écrits en java. Axis propose de simplifier au maximum les taches pour la création et l'utilisation des services web. Il permet notamment de générer automatiquement le fichier WSDL à partir d'une classe java et le code nécessaire à l'appel du service web.

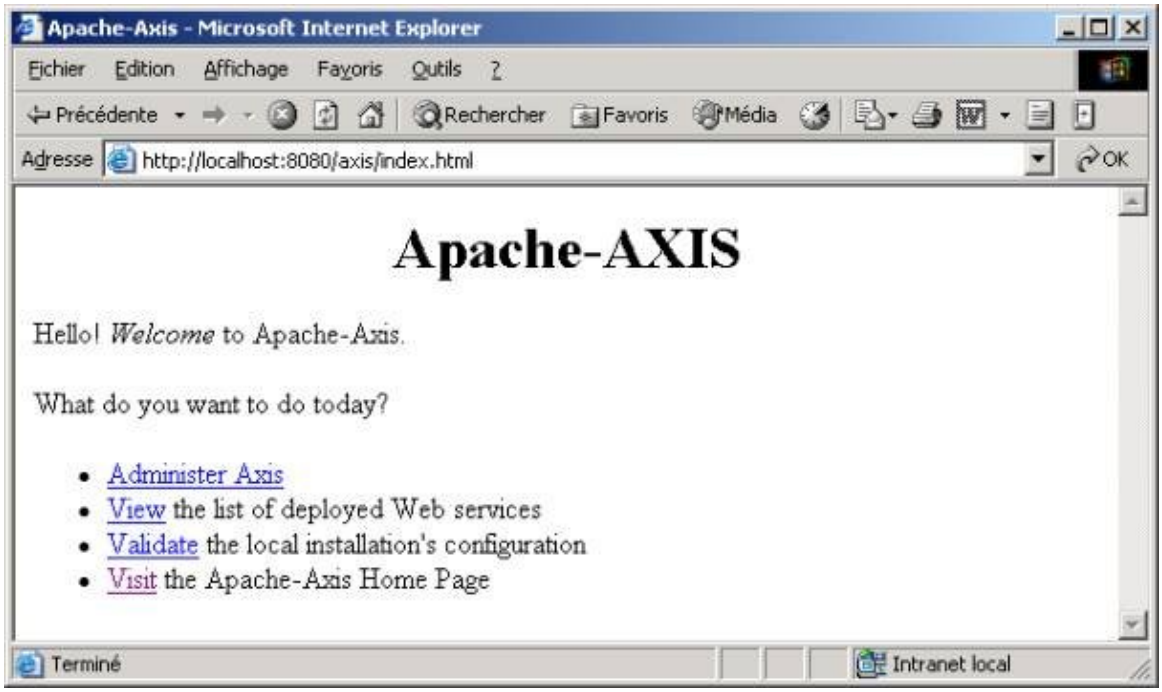
Pour son utilisation, Axis nécessite un J.D.K. 1.3 minimum et un conteneur de servlet (les exemples de cette section utilise Tomcat).

L'installation d'Axis est facile. Il faut télécharger la dernière version sur le site du groupe Apache : http://ws.apache.org/axis/releases.html

La version utilisée dans cette section est la 1.0. Il faut dézipper le fichier dans un répertoire (par exemple dans c:\java).

Il faut ensuite copier le répertoire axis dézippé dans le répertoire webapps de tomcat et lancer ou relancer Tomcat.

Pour tester si l'installation s'est déroulée correctement, il suffit de saisir l'url http://localhost:8080/axis/index.html dans un browser



Un clic sur le lien « Validate » permet d'exécuter une JSP qui fait un état des lieux de la configuration du conteneur et des API nécessaires et optionnelles accessibles.

Un clic sur le lien « View » permet de voir quels sont les services web qui sont installés.

Pour plus d'informations sur l'installation ou en cas de problème, il suffit de consulter la page correspondante sur le site d'Axis : http://ws.apache.org/axis/

Axis propose deux méthodes pour développer et déployer un service web :

- le déploiement automatique d'une classe java
- l'utilisation d'un fichier WSDD

44.4.1. Le déploiement automatique d'une classe java

Axis propose une solution pour facilement et automatiquement déployer une classe java en tant que service web. Il suffit simplement d'écrire la classe, de remplacer l'extension .java en .jws (java web service) et de copier le fichier dans le répertoire de la webapp axis.

44.4.2. L'utilisation d'un fichier WSDD

Cette solution est un peu moins facile à mettre en oeuvre mais elle permet d'avoir un meilleur contrôle sur le déploiement du service web.

Il faut écrire la classe java qui va contenir les traitements proposés par le service web.

Exemple :

```
public class MonServiceWebAxis2{
  public String message(String msg){
    return "Bonjour "+msg;
  }
}
```

Il faut compiler cette classe et mettre le fichier .class dans le répertoire WEB−INF/classes de la webapps axis.

Il faut créer le fichier WSDD qui va contenir la description du service web.

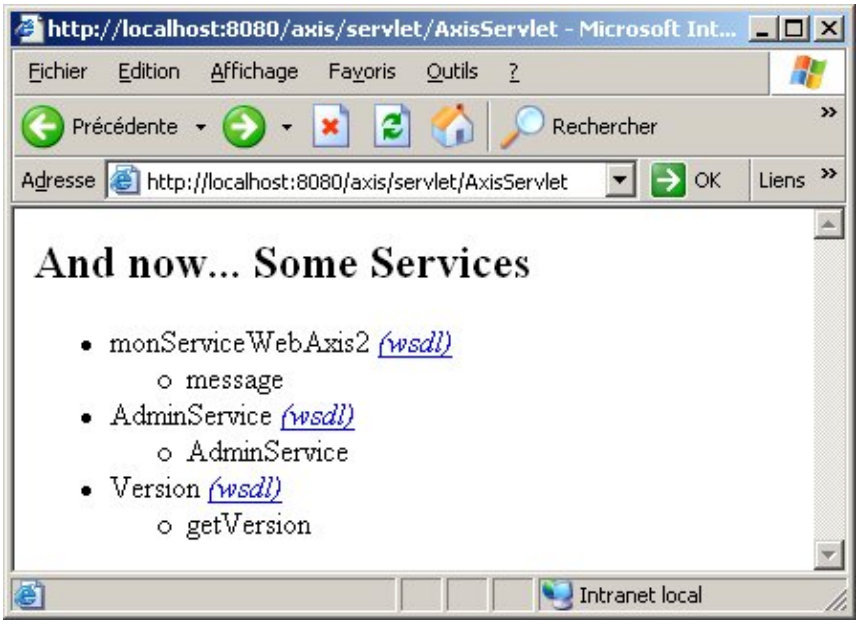
Exemple : deployMonServiceWebAxis2.wsdd

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="monServiceWebAxis2" provider="java:RPC">
           <parameter name="className" value="MonServiceWebAxis2"/>
    <parameter name="allowedMethods" value="*"/>
  </service>
</deployment>
```

Il faut ensuite deployer le service web en utilisant l'application AdminClient fournie par Axis.

Exemple : deployMonServiceWebAxis2.wsdd

```
C:\java\jwsdp−1.1\webapps\axis\WEB−INF\classes>java org.apache.axis.client.Admin
Client deployMonServiceWebAxis2.wsdd
− Processing file deployMonServiceWebAxis2.wsdd
− <Admin>Done processing</Admin>
```



44.4.3. L'utilisation d'un service web par un client

Pour faciliter l'utilisation d'un service web, Axis propose l'outil WSDL2Java qui génère automatiquement à partir d'un document WSDL des classes qui encapsulent l'appel à un service web. Grace à ces classes, l'appel d'un service web par un client ne nécessite que quelques lignes de code.

Exemple :

```
C:\java\jwsdp-1.1\webapps\axis\WEB-INF\classes>java org.apache.axis.wsdl.WSDL2Ja
va http://localhost:8080/axis/services/monServiceWebAxis2?wsdl
```

L'utilisation de l'outil WSDL2Java nécessite une url vers le document WSDL qui décrit le service web. Il génère à partir de ce fichier plusieurs classes dans le package localhost. Ces classes sont utilisées dans le client pour appeler le service web.

Exemple :

```
C:\java\jwsdp-1.1\webapps\axis\WEB-INF\classes>java org.apache.axis.wsdl.WSDL2Ja
va http://localhost:8080/axis/services/monServiceWebAxis2?wsdl
```

Il faut utiliser les classes générées pour appeler le service web.

Exemple :

```
import localhost.MonServiceWebAxis2;
import localhost.*;

public class MonServiceWebAxis2Client{

  public static void main(String[] args) throws Exception{
    MonServiceWebAxis2Service locator = new MonServiceWebAxis2ServiceLocator();
    MonServiceWebAxis2 monsw = locator.getmonServiceWebAxis2();
    String s = monsw.message("Jean Michel");
    System.out.println(s);
  }
}
```

Exécution :

```
C:\java\jwsdp-1.1\webapps\axis\WEB-INF\classes>javac MonServiceWebAxis2client.java
C:\java\jwsdp-1.1\webapps\axis\WEB-INF\classes>java MonServiceWebAxis2Client
Bonjour Jean Michel
```

Partie 5 : Les outils pour le développement

Le développement dans n'importe quel langage nécessite un ou plusieurs outils. D'ailleurs la multitude des technologies mises en oeuvre dans les projets récents nécessitent l'usage de nombreux outils.

Ce chapitre propose un recensement non exhaustif des outils utilisables pour le développement d'applications java et une présentation détaillée de certains d'entre eux.

Le JDK fournie un ensemble d'outils pour réaliser les développements mais leurs fonctionnalités se veulent volontairement limitées au strict miminum.

Enfin pour faciliter le développement d'applications, il est préférable d'utiliser une méthodologie pour l'analyse et d'utiliser ou de définir des normes lors du développement.

Cette partie contient plusieurs chapitres :

- Les outils du J.D.K. : indique comment utiliser les outils fournis avec le JDK
- Les outils libres et commerciaux : tente une énumération non exhaustive des outils libres et commerciaux pour utiliser java
- JavaDoc : explore l'outil de documentation fourni avec le JDK
- Java et UML : propose une présentation de la notation UML ainsi que sa mise en oeuvre avec Java
- Des normes de développement : propose de sensibiliser le lecteur à l'importance de la mise en place de normes de développement sur un projet et propose quelques règles pour définir une tel norme
- Les motifs de conception (design patterns) : présente certains modèles de conception en programmation orienté objet et leur mise en oeuvre avec Java
- Ant : propose une présentation et la mise en oeuvre de cet outil d'automatisation de la construction d'applications
- Maven : présente l'outil open source Maven qui facilite et automatise certaines taches de la gestion d'un projet

- Les frameworks de tests : propose une présentation de frameworks automatisant le passage de tests unitaires, notamment JUnit
- Des bibliothèques open source : présentation de quelques bibliothèques de la communauté open source particulièrement pratiques et utiles
- Des outils open source pour faciliter le développement : présentation de quelques outils de la communauté open source permettant de simplifier le travail des développeurs.

45. Les outils du J.D.K.

Le JDK de Sun fourni un ensemble d'outils qui permettent de réaliser des applications. Ces outils sont peu ergonomiques car ils s'utilisent en ligne de commande mais en contre partie ils peuvent toujours être utilisés.

Ce chapitre contient plusieurs sections :

45.1. Le compilateur javac

Cet outil est le compilateur : il utilise un fichier source java fourni en paramètre pour créer un ou plusieurs fichiers contenant le byte code Java correspondant. Pour chaque fichier source, un fichier portant portant le même nom avec l'extension .class est créé si la compilation se déroule bien. Il est possible qu'un ou plusieurs autres fichiers .class soit générés lors de la compilation de la classe si celle ci contient des classes internes. Dans ce cas, le nom du fichier des classes internes est de la forme classe$classe_interne.class. Un fichier .class supplémentaire est créé pour chaque classe interne.

45.1.1. La syntaxe de javac

La syntaxe est la suivante :

javac [options] [fichiers] [@fichiers]

Cet outil est disponible depuis le JDK 1.0

La commande attend au moins un nom de fichier contenant du code source java. Il peut y en avoir plusieurs, en les précisant un par un séparé par un escape ou en utilisant les jokers du système d'exploitation. Tous les fichiers précisés doivent obligatoirement posséder l'extension .java qui doit être précisée sur la ligne de commande.

Exemple : pour compiler le fichier MaClasse.

javac MaClasse.java

Exemple : pour compiler tous les fichier sources du répertoire

javac *.java

Le nom du fichier doit correspondre au nom de la classe contenue dans le fichier source. Il est obligatoire de respecter la casse du nom de la classe même sur des systèmes qui ne sont pas sensibles à la classe comme Windows.

Depuis le JDK 1.2, il est aussi possible de fournir un ou plusieurs fichiers qui contiennent une liste des fichiers à compiler. Chacun des fichiers à compiler doit être sur une ligne distincte. Sur la ligne de commande, les fichiers qui contiennent une liste doivent être précédés d'un caractère @

Exemple :

javac @liste

Contenu du fichier liste :
test1.java
test2.java

45.1.2. Les options de javac

Les principales options sont :

Option	Rôle
−classpath path	permet de préciser le chemin de recherche des classes nécessaires à la compilation
−d répertoire	les fichiers sont créés dans le répertoire indiqué. Par défaut, les fichiers sont créés dans le même répertoire que leurs sources.
−g	génère des informations débogage
−nowarn	le compilateur n'émet aucun message d'avertissement
−O	le compilateur procède à quelques optimisations. La taille du fichier généré peut augmenter. Il ne faut pas utiliser cette option avec l'option −g
−verbose	le compilateur affiche des informations sur les fichiers sources traités et les classes chargées
−deprecation	donne des informations sur les méthodes dépreciées qui sont utilisées

45.2. L'interpréteur java/javaw

Ces deux outils sont les interpréteurs de byte code : ils lancent le JRE, chargent les classes nécessaires et executent la méthode main de la classe.

java ouvre une console pour recevoir les messages de l'application alors que javaw n'en ouvre pas.

45.2.1. La syntaxe de l'outil java

java [options] classe [argument ...]
java [options] −jar fichier.jar [argument ...]
javaw [options] classe [argument ...]
javaw [options] −jar fichier.jar [argument ...]

classe être doit un fichier .class dont il ne faut pas préciser l'extension. La classe contenue dans ce fichier doit obligatoirement contenir une méthode main(). La casse du nom du fichier doit être respectée.

Cet outil est disponible depuis la version 1.0 du JDK.

Exemple:

java MaClasse

Il est possible de fournir des arguments à l'application.

45.2.2. Les options de l'outil java

Les principales options sont :

Option	Rôle
−jar archive	Permet d'exécuter une application contenue dans un fichier .jar. Depuis le JDK 1.2
−Dpropriete=valeur	Permet de définir une propriété système sous la forme propriete=valeur. propriété représente le nom de la propriété et valeur représente sa valeur. Il ne doit pas y avoir d'espace entre l'option et la définition ni dans la définition. Il faut utiliser autant d'option −D que de propriétés à définir. Depuis le JDK 1.1
−classpath chemins ou −cp chemins	permet d'indiquer les chemins de recherche des classes nécessaires à l'execution. Chaque répertoire doit être séparé avec un point virgule. Cette option utilisée annule l'utilisation de la variable système CLASSPATH
−classic	Permet de préciser que c'est la machine virtuelle classique qui doit être utilisée. Par défaut, c'est la machine virtuelle utilisant la technologie HotSpot qui est utilisée. Depuis le JDK 1.3
−version	Affiche des informations sur l'interpréteur
−verbose ou −v	Permet d'afficher chaque classe chargée par l'interpréteur
−X	Permet de préciser des paramètres particuliers à l'interpréteur. Depuis le JDK 1.2

L'option −jar permet d'executer une application incluse dans une archive jar. Dans ce cas, le fichier manifest de l'archive doit préciser qu'elle est la classe qui contient la méthode main().

45.3. L'outil JAR

JAR est le diminutif de Java ARchive. C'est un format de fichier qui permet de regrouper des fichiers contenant du byte−code Java (fichier .class) ou des données utilisées en temps que ressources (images, son, ...). Ce format est compatible avec le format ZIP : les fichiers contenus dans un jar sont compressés de façon indépendante du système d'exploitation.

Les jar sont utilisables depuis la version 1.1 du JDK.

45.3.1. L'intérêt du format jar

Leur utilisation est particulièrement pertinente avec les applets, les beans et même les applications. En fait, le format jar est le format de diffusion de composants java.

Les fichiers jar sont par défaut compressés ce qui est particulièrement intéressant quelque soit leurs utilisations.

Pour une applet, le browser n'effectue plus qu'une requête pour obtenir l'applet et ses ressources au lieu de plusieurs pour obtenir tous les fichiers nécessaires (fichiers .class, images, sons ...).

Un jar peut être signé ce qui permet d'assouplir et d'élargir le modèle de sécurité, notamment des applets qui ont des droits restreints par défaut.

Les beans doivent obligatoirement être diffusés sous ce format.

Les applications sous forme de jar peuvent être exécutées automatiquement.

Une archive jar contient un fichier manifest qui permet de préciser le contenu du jar et de fournir des informations sur celui ci (classe principale, type de composants, signature ...).

45.3.2. La syntaxe de l'outil jar

Le JDK fourni un outil pour créer des archives jar : jar. C'est un outil utilisable avec la ligne de commandes comme tous les outils du JDK.

La syntaxe est la suivante :

jar [option [jar [manifest [fichier

Cet outil est disponible depuis la version 1.1 du JDK.

Les options sont :

Option	Rôle
c	Création d'un nouvelle archive
t	Affiche le contenu de l'archive sur la sortie standard
x	Extraction du contenu de l'archive
u	Mise à jour ou ajout de fichiers à l'archive : à partir de java 1.2
f	Indique que le nom du fichier contenant l'archive est fourni en paramètre
m	Indique que le fichier manifest est fourni en paramètre
v	Mode verbeux pour avoir des informations complémentaires
0 (zéro)	Empêche la compression à la création
M	Empêche la création automatique du fichier manifest

Pour fournir des options à l'outil jar, il faut les saisir sans '−' et les accoler les uns aux autres. Leur ordre n'a pas d'importance.

Une restriction importante concerne l'utilisation simultanée du paramère 'm' et 'f' qui nécessite respectivement le nom du fichier manifest et le nom du fichier archive en paramètre de la commande. L'ordre de ces deux paramètres doit être identique à l'ordre des paramètres 'm' et 'f' sinon une exception est levée lors de l'exécution de la commande

Exemple (code Java 1.1) :

```
C:\jumbo\Java\xagbuilder>jar cmf test.jar manif.mf *.class
java.io.IOException: invalid header field
        at java.util.jar.Attributes.read(Attributes.java:354)
        at java.util.jar.Manifest.read(Manifest.java:161)
        at java.util.jar.Manifest.<init>(Manifest.java:56)
        at sun.tools.jar.Main.run(Main.java:125)
        at sun.tools.jar.Main.main(Main.java:904)
```

Voici quelques exemples de l'utilisation courante de l'outil jar :

- Création d'un jar avec un fichier manifest créé automatiquement contenant tout les fichiers .class du répertoire courant

jar cf test.jar *.class

- lister le contenu d'un jar

jar tf test.jar

- Extraire le contenu d'une archive

jar xf test.jar

45.3.3. La création d'une archive jar

L'option 'c' permet de créer une archive jar. Par défaut, le fichier créé est envoyé sur la sortie standard sauf si l'option 'f' est utilisée. Elle précise que le nom du fichier est fourni en paramètre. Par convention, ce fichier a pour extension .jar.

Si le fichier manifest n'est pas fourni, un fichier est créé par défaut dans l'archive jar dans le répertoire META−INF sous le nom MANIFEST.MF

Exemple (code Java 1.1) : Création d'un jar avec un fichier manifest créé automatiquement contenant tout les fichiers .class du répertoire courant

```
jar cf test.jar *.class
```

Il est possible d'ajouter des fichiers contenus dans des sous répertoires du répertoire courant : dans ce cas, l'arborescence des fichiers est conservée dans l'archive.

Exemple (code Java 1.1) : Création d'un jar avec un fichier manifest fourni contenant tous les fichiers .class du répertoire courant et tous les fichiers du répertoire images

```
jar cfm test.jar manifest.mf .class images
```

Exemple (code Java 1.1) : Création d'un jar avec un fichier manifest fourni contenant tous les fichiers .class du répertoire courant et tous les fichiers .gif du répertoire images

```
jar cfm test.jar manifest.mf *.class images/*.gif
```

45.3.4. Lister le contenu d'une archive jar

L'option 't' permet de donner le contenu d'une archive jar.

Exemple (code Java 1.1) : lister le contenu d'une archive jar

```
jar tf test.jar
```

Le séparateur des chemins des fichiers est toujours un slash quelque soit la plate−forme car le format jar est indépendant de toute plate−forme. Les chemins sont toujours donnés dans un format relatif et non pas absolu : le chemin est donné par rapport au répertoire courant. Il faut en tenir compte lors d'une extraction.

Exemple (code Java 1.1) :

```
C:\jumbo\bin\test\java>jar tvf test.jar
  2156 Thu Mar 30 18:10:34 CEST 2000 META−INF/MANIFEST.MF
   678 Thu Mar 23 12:30:00 CET 2000   BDD_confirm$1.class
   678 Thu Mar 23 12:30:00 CET 2000   BDD_confirm$2.class
  4635 Thu Mar 23 12:30:00 CET 2000   BDD_confirm.class
   658 Thu Mar 23 13:18:00 CET 2000   BDD_demande$1.class
   657 Thu Mar 23 13:18:00 CET 2000   BDD_demande$2.class
   662 Thu Mar 23 13:18:00 CET 2000   BDD_demande$3.class
```

```
  658 Thu Mar 23 13:18:00 CET 2000   BDD_demande$4.class
 5238 Thu Mar 23 13:18:00 CET 2000   BDD_demande.class
  649 Thu Mar 23 12:31:28 CET 2000   BDD_resultat$1.class
 4138 Thu Mar 23 12:31:28 CET 2000   BDD_resultat.class
  533 Thu Mar 23 13:38:28 CET 2000   Frame1$1.class
  569 Thu Mar 23 13:38:28 CET 2000   Frame1$2.class
  569 Thu Mar 23 13:38:28 CET 2000   Frame1$3.class
 2150 Thu Mar 23 13:38:28 CET 2000   Frame1.class
  919 Thu Mar 23 12:29:56 CET 2000   Test2.class
```

45.3.5. L'extraction du contenu d'une archive jar

L'option 'x' permet d'extraire par défaut tous les fichiers contenus dans l'archive dans le répertoire courant en respectant l'arborescence de l'archive. Pour n'extraire que certains fichiers de l'archive, il suffit de les préciser en tant que paramètres de l'outil jar en les séparant par un espace. Pour une extraction totale ou partielle de l'archive, les fichiers sont extraits en conservant la hiérarchie des répertoires qui les contiennent.

Exemple (code Java 1.1) : Extraire le contenu d'une archive

```
jar xf test.jar
```

Exemple (code Java 1.1) : Extraire les fichiers test1.class et test2.class d'une archive

```
jar xf test.jar test1.class test2.class
```

Attention : lors de l'extraction, l'outil jar écrase tous les fichiers existants sans demander de confirmation.

45.3.6. L'utilisation des archives jar

Dans une page HTML, pour utiliser une applet fournie sous forme de jar, il faut utiliser l'option archive du tag applet. Cette option attend en paramètre le fichier jar et son chemin relatif par rapport au répertoire contenant le fichier HTML.

Exemple (code Java 1.1) : le fichier HTML et le fichier MonApplet.jar sont dans le même répertoire

```
<applet code=MonApplet.class
        archive="MonApplet.jar"
        width=300 height=200>
</applet>
```

Avec java 1.1, l'exécution d'une application sous forme de jar se fait grâce au jre. Il faut fournir dans ce cas le nom du fichier jar et le nom de la classe principale.

Exemple (code Java 1.1) :

```
jre -cp MonApplication.jar ClassePrincipale
```

Avec java 1.2, l'exécution d'une application sous forme de jar impose de définir la classe principale (celle qui contient la méthode main) dans l'option Main-Class du fichier manifest. Avec cette condition l'option -jar de la commande java permet d'exécuter l'application.

Exemple (code Java 1.2) :

```
java -jar MonApplication.jar
```

45.3.7. Le fichier manifest

Le fichier manifest contient de nombreuses informations sur l'archive et son contenu. Ce fichier est le support de toutes les fonctionnalités particulières qui peuvent être mise en oeuvre avec une archive jar.

Dans une archive jar, il ne peut y avoir qu'un seul fichier manifest nommé MANIFEST dans le répertoire META−INF de l'archive.

Le format de ce fichier est de la forme clé/valeur. Il faut mettre un ':' et un espace entre la clé et la valeur.

```
C:\jumbo\bin\test\java>jar xf test.jar META-INF/MANIFEST.MF
```

Cela créé un répertoire META−INF dans le répertoire courant contenant le fichier MANIFEST.MF

Exemple (code Java 1.1) :

```
Manifest-Version: 1.0
Name: BDD_confirm$1.class
Digest-Algorithms: SHA MD5
SHA-Digest: ntbIs5E5YNilE4mf570JoIF9akU=
MD5-Digest: R3zH0+m9lTFq+BlQvfQdHA==
Name: BDD_confirm$2.class
Digest-Algorithms: SHA MD5
SHA-Digest: 3QEF8/zmiTAP7MHFPU5wZyg9uxc=
MD5-Digest: swBXXptrLLwPMw/bpt6F0Q==
Name: BDD_confirm.class
Digest-Algorithms: SHA MD5
SHA-Digest: pZBT/o8YeDG4q+XrHRgrB08k4HY=
MD5-Digest: VFvY4sGRfjV1ciM9C+QIdg==
```

Dans le fichier manifest créé automatiquement avec le JDK 1.1, chaque fichier possède au moins une entrée de type 'Name' et des informations les concernant.

Entre les données de deux fichiers, il y a une ligne blanche.

Dans le fichier manifest créé automatiquement avec le JDK 1.2, il n'y a plus d'entrée pour chaque fichier.

Exemple (code Java 1.1) :

```
Manifest-Version: 1.0
Created-By: 1.3.0 (Sun Microsystems Inc.)
```

Le fichier manifest généré automatiquement convient parfaitement si l'archive est utilisée uniquement pour regrouper les fichiers. Pour une utilisation plus spécifique, il faut modifier ce fichier pour ajouter les informations utiles.

Par exemple, pour une application exécutable (à partir de java 1.2) il faut ajouter une clé Main−Class en lui associant le nom de la classe dans l'archive qui contient la méthode main.

45.3.8. La signature d'une archive jar

La signature d'une archive jar joue un rôle important dans les processus de sécurité de java. La signature d'une archive permet à celui qui utilise cette archive de lui donner des droits étendus une fois que la signature a été reconnue.

Avec Java 1.1 une archive signée possède tous les droits.

Avec Java 1.2 une archive signée peut se voir attribuer des droits particuliers définis un fichier policy.

45.4. Pour tester les applets : l'outil appletviewer

Cet outil permet de tester une applet. L'intérêt de cet outil est qu'il permet de tester une applet avec la version courante du JDK. Un navigateur classique nécessite un plug−in pour utiliser une version particulière du JRE. Cet outil est disponible depuis la version 1.0 du JDK.

En contre partie, l'appletviewer n'est pas prévu pour tester les pages HTML. Il charge une page HTML fournie en paramètre, l'analyse, charge l'applet qu'elle contient et execute cet applet.

La syntaxe est la suivante : appletviewer [option] fichier

L'appletview recherche le tag HTML <APPLET>. A partir du JDK 1.2, il recherche aussi les tags HTML <EMBED> et <OBJECT>.

Il possède plusieurs options dont les principales sont :

Option	Rôle
−J	Permet de passer un paramètre à la JVM. Pour passer plusieurs paramètres, il faut utiliser plusieurs options −J. Depuis le JDK 1.1
−encoding	Permet de préciser le jeu de caractères de la page HTML

L'appletviewer ouvre une fenêtre qui possède un menu avec les options suivantes :

Option de menu	Rôle
Restart	Permet d'arrêter et de redémarrer l'applet
Reload	Permet d'arrêter et de recharger l'applet
Stop	Permet d'arrêter l'execution de l'applet. Depuis le JDK 1.1
Save	Permet de sauvegarder l'applet en la serialisant dans un fichier applet.ser. Il est necessaire d'arrêter l'applet avant d'utiliser cet option. Depuis le JDK 1.1
Start	Permet de démarrer l'applet. Depuis le JDK 1.1
Info	Permet d'afficher les informations de l'applet dans une boite de dialogue. Ces informations sont obtenues par les méthodes getAppletInfo() et getParameterInfo() de l'applet.
Print	Permet d'imprimer l'applet. Depuis le JDK 1.1
Close	Permet de fermer la fenêtre courante
Quit	Permet de fermer toutes les fenêtres ouvertes par l'appletviewer

45.5. Pour générer la documentation : l'outil javadoc

Cet outil permet de générer une documentation à partir des données insérées dans le code source.

45.5.1. La syntaxe de javadoc

La syntaxe est la suivante :

javac [options] [fichiers] [@fichiers]

Cet outil est disponible depuis le JDK 1.0

45.5.2. Les options de javadoc

Les principales options sont :

Option	Rôle
−1.1	Permet de générer une documentation au format défini par le JDK 1.1
−author	Permet d'inclure dans la documentation les données du tag @author
−d	Permet de préciser le répertoire qui va contenir les fichiers générés
−nodeprecated	Permet d'omettre les informations sur les éléments deprecated
−nodeprecatedlist	Permet de ne pas générer la page contenant les éléments deprecated
−noindex	Permet de ne pas générer la page index
−notree	Permet de ne pas générer la page contenant la hiérarchie de classes
−private	Permet d'inclure les éléments déclarés private
−protected	Permet d'inclure les éléments déclarés protected
−public	Permet de n'inclure que les éléments déclarés public
−verbose	Permet de fournir des informations lors de la génération

Des informations supplémentaires sur les éléments à inclure dans le code source sont fournies dans le chapitre consacré à Javadoc

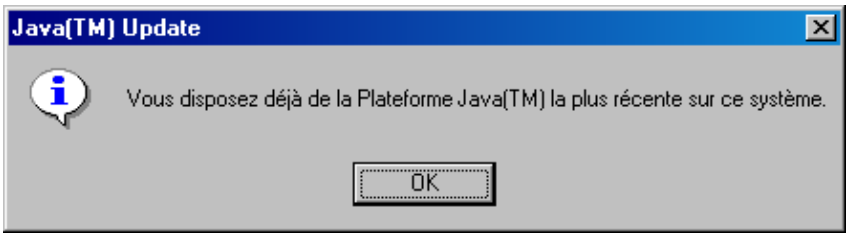
45.6. Java Check Update

Jucheck (Java Update Check) est un outil proposé par Sun pour permettre une mise à jour automatique de l'environnement d'exécution Java.

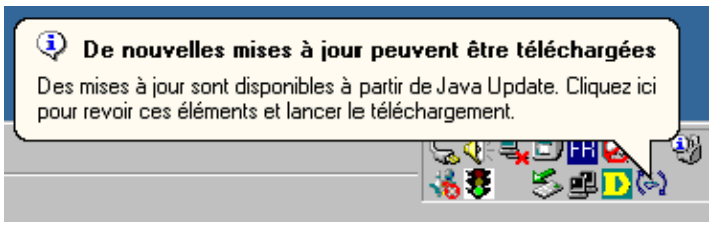
L'outil jusched.exe est installé par défaut et configuré pour une exécution automatique depuis la version 1.4.2 du J2SE. Il permet une automatisation de l'exécution de jucheck.exe.

Pour lancer manuellement la mise à jour, il suffit d'exécuter le programme jucheck.exe dans le répertoire bin du JRE.

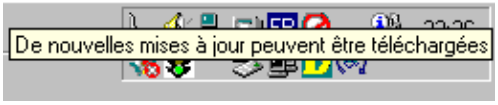
Si aucune mise à jour n'est disponible, un message est affiché :



Sinon une bulle d'aide informe que des mises à jour peuvent être téléchargées.



En laissant le curseur de la souris sur l'icône du programme de mise à jour, une bulle d'aide est affichée.

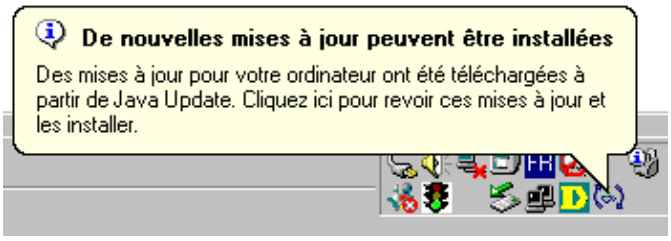


Pour télécharger les mises à jour, il suffit d'utiliser l'option « Télécharger » du menu contextuel associé à l'icône

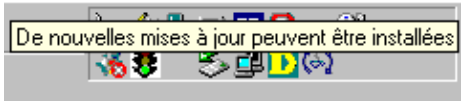
Une boîte de dialogue permet de demander le téléchargement des éléments dont la version est indiquée.

Cliquez sur le bouton « Téléchargement ».

Une fois le téléchargement terminée, une bulle est affichée.



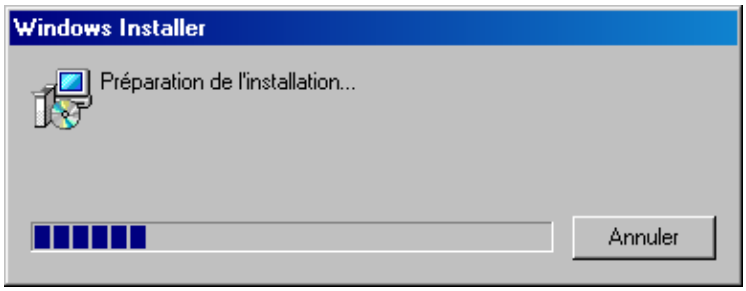
En laissant le curseur de la souris sur l'icône du programme de mise à jour, une bulle d'aide est affichée.



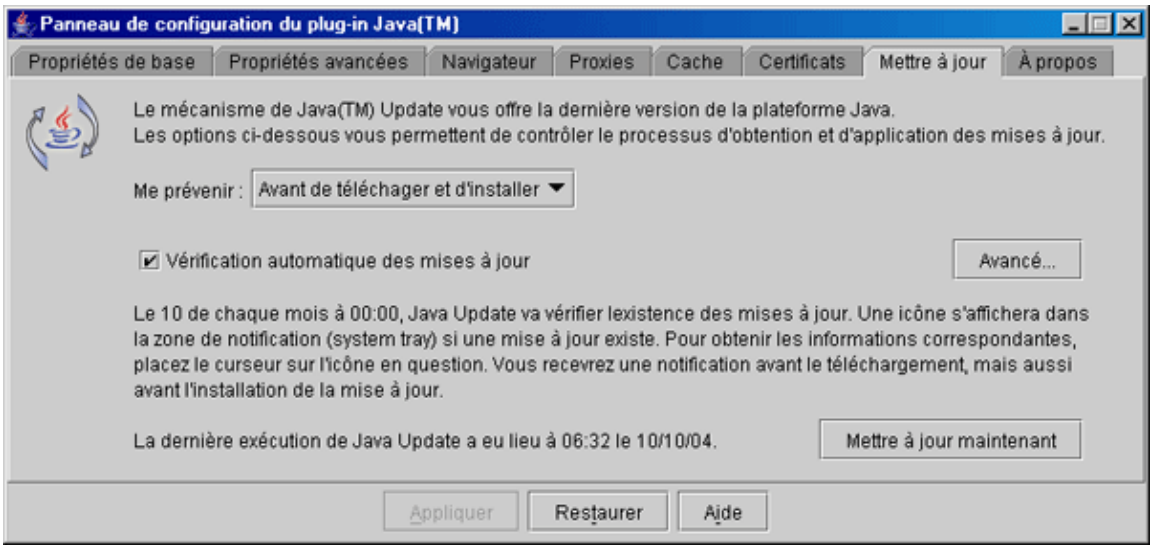
Pour installer les mises à jour, il suffit d'utiliser l'option « Installer » du menu contextuel associé à l'icône

Cliquez sur le bouton « Installer ».

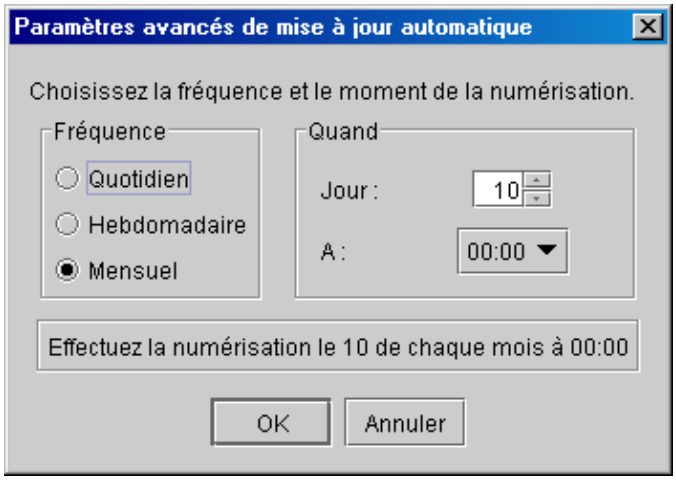
L'assistant se lance pour diriger les différentes étapes



L'option « Propriétés » permet d'ouvrir une boîte de dialogue pour gérer les paramètres des mises à jour dans la l'onglet « Mettre à jour ».



Le bouton « Avancé » permet de définir les paramètres de recherche automatique.


Paramètres avancés de mise à jour automatique
Choisissez la fréquence et le moment de la numérisation.
Fréquence
Quotidien
Hebdomadaire
Mensuel
Quand
Jour :
10
A :
00:00
Effectuez la numérisation le 10 de chaque mois à 00:00
OK
Annuler

46. Les outils libres et commerciaux

Pour développer des composants en java (applications clientes, applets, applications web, services web ...), il existe une large gamme d'outils commerciaux et libres pour répondre à ce vaste marché.

Comme dans d'autres domaines, les avantages et les inconvénients de ces outils sont semblables selon leur catégorie bien qu'ils ne puissent pas être complétement généralisés :

	Avantages	Inconvénient
Outils commerciaux	une meilleur ergonomie une hot line dédiée	le prix
Outils libres	la gratuité des mises à jour fréquentes (variable selon le projet)	pas de support officiel (aide communautaire via les forums)

Certains de ces outils libres n'ont que peu de choses à envier à certains de leurs homologues commerciaux : ainsi Tomcat du projet Jakarta est l'implémentation de référence pour ce qui concerne les servlets et les JSP.

Enfin certains éditeurs, surtout dans le domaine des IDE, proposent souvent une version limitée (dans les fonctionnalités ou dans le temps)) mais gratuite qui permet d'utiliser et d'évaluer le produit.

L'évolution des ces outils suit l'évolution du marché concernant java : développement d'applet (web client), d'application automone et C/S, et maintenant développement côté serveur (applications et services web).

La liste des produits de ce chapitre est loin d'être exhaustive mais représente les plus connus ou ceux que j'utilise.

Ce chapitre contient plusieurs sections :

46.1. Les environnements de développements intégrés (IDE)

Les environnements de développements intégrés regroupent dans un même outil la possibilité d'écrire du code source, de concevoir une application de façon visuelle par assemblage de beans, d'exécuter et de débugger le code.

D'une façon générale, ils sont tous très gourmands en ressources machines : un processeur rapide, 256 Mo de RAM pour être à l'aise ... En fait la plupart de ces outils sont partiellement ou totalement écrits en Java.

Le choix d'un IDE doit tenir compte de plusieurs caracéristiques : ergonomie et convivialité pour faciliter l'utilisation, fonctionnalités de bases et avancées pour accroître la productivité, robustesse, support des standards, ... Tous les éditeurs

proposent une version libre qui permet d'évaluer leur produit.

46.1.1. Borland JBuilder

Borland est spécialisé depuis des années dans la création d'outils de développement possédant une excellente réputation. Ainsi Jbuilder est un IDE ergonomique qui génère un code "propre", ce qui lui vaut d'être l'IDE java le plus utilisé dans le monde. Depuis sa version 3.5, JBuilder est écrit en Java ce qui lui permet de s'exécuter sans difficulté sur plusieurs plateformes notamment Windows, Linux ou Solaris.

http://www.borland.fr/jbuilder/index.html

Le produit dispose de nombreuses caractéristiques qui facilitent le travail du développeur : la technologie CodeInsight facilite grandement l'écriture du code dans l'éditeur, de nombreux assistants facilitent la génération de code ...

Il existe plusieurs éditions :

- foundation ou édition personnelle (depuis la version 5) :
- professionnelle
- entreprise

La version 5 intègre fortement XML et elle est très ouverte : par exemple la version entreprise intègre des outils libres (Xerces, Xalan, Tomcat), permet la création d'archive jar, war et ear, permet le déploiement d'application J2EE vers des serveurs d'application concurent tel que Weblogic ou Websphere et des applications avec java web start, peut travailler avec les principaux gestionnaires de versions (CSV, Source Safe, Clear Case) ...

La version 9.0 de cet outil est publiée en 2003.

46.1.2. IBM Visual Age for Java

IBM propose une famille d'outils pour le développement avec différents langages dont une version dédiée à java.

Visual Age for Java (VAJ) est un outil novateur dans son ergonomie et son utilisation qui sont complètement différentes des autres EDI. Les débuts de son utilisation sont parfois déroutant mais une persévérence permet de révéler toute sa puissance.

http://www-4.ibm.com/software/ad/vajava/

La fenêtre principale (plan de travail) est séparée en deux parties :

- l'espace de travail : il contient et organise les différents éléments (projets, packages, classes, méthodes ...)
- le code source : si l'élement sélectionné dans l'espace de travail contient du code, il est visualisé et modifiable dans cette partie

Par défaut le code est éditable par méthode mais depuis la version 3.5, il est toutefois possible de visualiser le code source complet mais les opérations réalisables dans ce mode sont moins nombreuses.

VAJ possède plusieurs points forts : le regroupement de toutes les classes et leur organisation dans l'espace de travail, la compilation incrémentale à l'écriture et au débuggage, le travail collaboratif avec le contrôle de version dans un référentiel (repository). Tous ces points facilitent le développement de gros projets mais il n'est pas adapté pour le déploiement du code écrit : il faut utiliser un autre produit.

VAJ est un outil puissant particulièrement adapté aux utilisateurs chevronnés pour de gros projets.

VAJ n'est plus supporté par IBM : il est remplacé par la famille d'outils Websphere Studio Application Developper.

46.1.3. IBM Websphere Studio Application Developper

Websphere Studio Application Developper (WSAD) représente le nouvel outil de développement d'application Java/web d'IBM. Il représente une fusion de nombreuses fonctionnalités des outils Visual Age for Java et Websphere Studio. Le coeur de l'outil est composé par Websphere Studio Workbench dont une partie du code a été fournie à la communauté open source pour devenir le projet Eclipse. Le but est de fournir un framework commun pour un outil de développement modulaire.

http://www−4.ibm.com/software/ad/studioappdev/

L'avantage de cette modularité est de fournir dans un même outil des fonctionnalités qui nécessitaient jusqu'à présent l'usage de plusieurs outils dont l'inter−opérabilité n'était pas parfaite.

Pour le moment, WSAD version 4.0 est orienté developpement Java/web : il ne permet pas de développement d'applications graphiques en mode RAD.

46.1.4. Netbeans

Netbeans est un environnement de développement en java open source écrit en java. Le produit est composé d'une partie centrale à laquelle il est possible d'ajouter des modules tel que Poseidon pour la création avec UML.

http://www.netbeans.org/

Netbeans est la base de l'IDE de Sun : Forte for Java.

46.1.5. Sun Forte for java

Forte for java est basé sur Netbeans que Sun a racheté.

http://www.sun.com/forte/ffj

La version Community Edition de Forte for java est téléchargeable gratuitement.

46.1.6. JCreator

http://www.jcreator.com

Jcreator existe en deux version : la version "LE" est en freeware et la version "PRO" est en shareware. Il est particulièrement rapide car il est écrit en code natif.

46.1.7. Le projet Eclipse

Eclipse est un projet open source à l'origine développé par IBM pour ses futurs outils de développement et offert à la communauté. Le but est de fournir un outil modulaire capable non seulement de faire du développement en java mais aussi dans d'autres langage et d'autres activités. Cette polyvalence est liée au développement de modules réalisés par la communauté ou des entités commerciales.

http://www.eclipse.org/

L'espace de travail permet de voir des perspectives qui assurent une vision particulière d'un projet. Chaques perspectives contient des vues et des éditeurs qui pemettent de travailler sur une entité.

La version 2.1 de cet outil est publiée en 2003.

46.1.8. Webgain Visual Café

Webgain Studio propose un ensemble d'outils (Visual Café, Dreamweaver Ultradev, Top link, Structure Builder, Weblogic) pour la création d'applications e−business. Visual Café est l'IDE de développement en java.

http://www.webgain.com/

Visual Café existe en trois version : standard, expert et entreprise suite.

Il permet de travailler avec plusieurs JDK : 1.1, 1.2 et 1.3

Malheureusement cet outil n'est plus disponible.

46.1.9. Omnicore CodeGuide

Omnicore Sofware propose un IDE nommé CodeGuide.

http://www.omnicore.com/

La version 6.0 de cet outil est publiée en 2003.

46.1.10. IntelliJ IDEA

http://www.intellij.com/idea/

46.2. Les serveurs d'application

Les serveurs d'applications sont des outils qui permettent l'exécution de composants Java côté serveur (servlets, JSP, EJB, ...).

46.2.1. IBM Websphere Application Server

Websphere Application Server (WAS) est le serveur d'application de la famille d'outils Websphere. Il permet le déploiement de composants Java orienté entreprise

http://www−4.ibm.com/software/webservers/

La version 4 est certifiée J2EE 1.2. Elle permet la mise en oeuvre des servlets, JSP, EJB et services Web (SOAP, UDDI, WSDL, XML). Cette version est proposée en 4 éditions qui supporte tout ou partie de ces composants :

- Standard Edition : pour les applications web utilisant des serlets, des JSP et XML
- Advanced Edition: supporte en plus les EJB, la répartition de charges sur plusieurs machines
- Advanced Single Server Edition : supporte toute les API J2EE mais uniquement sur une seule machine. Cette version ne peut pas être utilisée en production.

- Enterprise Edition : supporte en plus CORBA et la connection aux ressources de l'entreprise

La version 5 est certifiée J2EE 1.3.

46.2.2. BEA Weblogic

Weblogic est une famille de produit proposéBEA dont Weblogic Server qui est un des leader mondial des serveurs d'applications.

http://fr.bea.com/produits/index.jsp

La version 8.1 de cet outil est publiée en 2003.

46.2.3. iplanet / Sun One

http://www.iplanet.com/

46.2.4. Borland Entreprise Server

http://www.borland.fr/besappserver/index.html

46.2.5. Macromedia JRun

JRun est l'implémentation d'un serveur d'applications de Macromedia.

http://www.macromedia.com/software/jrun/

La version 4 est certifié J2EE 1.3.

46.2.6. Oracle 9i Application Server

Oracle propose un serveur d'applications certifié J2EE 1.3

http://www.oracle.com/ip/deploy/ias/

46.3. Les conteneurs web

Les conteneurs web sont des applications qui permettent d'exécuter du code Java utilisé pour définir des servlets et des JSP.

46.3.1. Apache Tomcat

Tomcat est un conteneur d'applications web (servlets et JSP) développé par la fondation Apache. C'est l'implémentation de référence pour les API servlets et JSP : il est donc pleinement compatible avec les spécifications J2EE de ces API.

http://jakarta.apache.org/tomcat/

Les API supportés dépendent de la version du produit :

Version	API Servlets	API JSP
3.0, 3.1, 3.2, 3.3	2.2	1.1
4.0, 4.1	2.3	1.2
5.0	2.4	2.0

46.3.2. Caucho Resin

Resin est un moteur de servlet et de JSP qui intègre un serveur web.

http://www.caucho.com/

46.3.3. Enhydra

Enhydra est un projet open source, initialement créer par Lutris technologies , pour développer un conteneur web pour Servlets et JSP. Il fourni en plus quelques fonctionnalités supplémentaires pour utiliser XML, mapper des données avec des objets et gérer un pool de connection vers des bases de données

http://enhydra.enhydra.org/

46.4. Les conteneurs d'EJB

Les conteneurs d'EJB sont des applications qui fournissent un environnement d'execution pour les EJB.

46.4.1. JBoss

JBoss est un projet open source développé en java pour fournir un environnement d'exécution d'EJB respectant les spécification J2EE.

http://www.jboss.org

JBoss est composé d'un ensemble d'outils : JBoss Server, JBoss MQ (implémentation de JMS), JBoss MX, JBoss TX (implémentation de JTA/JTS), JBoss SX , JBoss CX et JBoss CMP.

46.4.2. Jonas

JOnAS est un projet open source développé en Java visant a réaliser une implémentation des spécifications EJB 1.1, JTA 1.0.1, JDBC 2.0 and JMS 2.0.1.

http://www.objectweb.org/jonas/index.html

46.4.3. OpenEJB

OpenEJB est un projet open source pour developper un conteneur d'EJB qui respecte les spécifications 2.0 des EJB. Pour le moment, le projet est en cours de développement et il n'existe pas encore de binaires (seuls les sources sont disponibles).

http://openejb.exolab.org/

46.5. Les outils divers

46.5.1. Jikes

Jikes est un compilateur Java open source écrit par IBM en code natif pour Windows et Linux. Son exécution est donc extrèmement rapide d'autant plus lorsqu'il s'agit de très gros projets sur une machine peu véloce.

http://www10.software.ibm.com/developerworks/opensource/jikes/

Pour utiliser Jikes, il sufiit de décompresser l'archive et de mettre le fichier exécutable dans un répertoire inclus dans le CLASSPATH. Enfin, il faut déclarer une variable système JIKESPATH qui doit contenir les différents répertoires contenant les classes et les jar notamment le fichier rt.jar du JRE.

46.5.2. GNU Compiler for Java

GCJ fait parti du projet GCC (GNU Compiler Collection). Le projet GCC propose un compilateur pour plusieurs langages (C, C++, Objective C, Java ...) permettant de produire un exécutable pour plusieurs plate−formes.

GCJ est donc un font−end pour utiliser GCC à partir de code Java. Il permet notamment de :

- de compiler du code source Java en byte−code
- de compiler du code source Java en un exécutable contenant du code machine dépendant d'un système d'exploitation

Pour un exécutable, le fichier final est lié avec une bibliothèque dédiée nommée libgcj qui contient entre autre les classes de bases et le ramasse miette.

La plupart des API de la plate−forme Java 2 sont supportées à l'exception notable de la bibliothèque AWT. Pour obtenir plus d'information sur la compatibilité, il suffit de consulter la page http://gcc.gnu.org/java/status.html

Son utilisation sous Windows nécessite un environnement particulier : CygWin ou MinGW (ce dernier étant retenu dans la suite de cette section).

Téléchargez sur le site http://www.mingw.org/download.shtml les fichiers : MinGW−3.1.0−1.exe (14,5 Mo) et MSYS−1.0.9.exe (2,7 Mo). (les noms de fichiers indiqués correspondent à la version courante au moment de l'écriture de cette section).

Lancez le programme MinGW−3.1.0−1.exe

Le programme d'installation se lance et demande une confirmation de l'installation : cliquer sur « Oui ». Un assistant permet de guider les différentes étapes de l'installation :

- Cliquez sur « Next »
- Lisez la licence et cliquez sur « Yes » si vous l'acceptez
- Lisez les informations et cliquez sur « Next »
- Choisissez le répertoire d'installation et cliquez sur « Next » (pour la suite des instruction, le répertoire par défaut c:\MinGW est utilisé)
- Cliquez sur « Install »
- Une fois l'installation terminée, cliquez sur « Finish »

Lancez le programme MSYS−1.0.9.exe

Le programme d'installation se lance et demande une confirmation de l'installation : cliquer sur « Oui ». Un assistant permet de guider les différentes étapes de l'installation :

- Cliquez sur « Next »
- Lisez la licence et cliquez sur « Yes » si vous l'acceptez
- Lisez les informations et cliquez sur « Next »
- Choisissez le répertoire d'installation et cliquez sur « Next » (pour la suite des instruction, le répertoire C:\MinGW\msys\1.0 est utilisé)
- Sélectionnez l'unique composant à installer et cliquez sur « Next »
- Sélectionnez le raccourci dans le menu Programme (MinGW par défaut) et cliquez sur « Next »
- Cliquez sur « Install »
- L'installation s'exécute et lance un script dos de configuration : il suffit de répondre aux questions

Exemple :

```
C:\MinGW\msys\1.0\postinstall>..\bin\sh.exe pi.sh
This is a post install process that will try to normalize between
your MinGW install if any as well as your previous MSYS installs
if any.  I don't have any traps as aborts will not hurt anything.
Do you wish to continue with the post install? [yn ] y
Do you have MinGW installed? [yn ] y
Please answer the following in the form of c:/foo/bar.
Where is your MinGW installation? c:/Mingw
Creating /etc/fstab with mingw mount bindings.
        Normalizing your MSYS environment.
You have script /bin/awk
You have script /bin/cmd
You have script /bin/echo
You have script /bin/egrep
You have script /bin/ex
You have script /bin/fgrep
You have script /bin/printf
You have script /bin/pwd
You have script /bin/rvi
You have script /bin/rview
You have script /bin/rvim
You have script /bin/vi
You have script /bin/view
Oh joy, you do not have c:/Mingw/bin/make.exe. Keep it that way.
C:\MinGW\msys\1.0\postinstall>pause
Appuyez sur une touche pour continuer...
```

- Appuyer sur une touche pour fermer la boîte dos
- Une fois l'installation terminée, cliquez sur « Finish »

Remarque : il est fortement recommandé de ne pas utiliser d'espace dans les noms des répertoires d'installation de MinGW et de MSYS.

Il faut pour plus de facilité d'utilisation ajouter à la variable PATH de l'environnement système les répertoires C:\MinGW\bin et C:\MinGW\msys\1.0\bin.

La version de GCC founie avec MinGW précédemment installé est la 3.2. Pour utiliser GCJ, il faut utiliser la 3.3 et donc opérer une mise à jour.

Il faut télécharger les fichiers gcc−core−3.3.1−20030804−1.tar.gz, gcc−g++−3.3.1−20030804−1.tar.gz et gcc−java−3.3.1−20030804−1.tar.gz, les décompresser et extraire l'image tar dans le répertoire c:\MinGW.

Remarque : en standard aucun outil ne permet de traiter des fichiers gz et tar. Il faut utiliser un outil tiers.

Pour s'assurer de la bonne installation, il suffit d'ouvrir une boite Dos et d'exécuter la commande gcj. Le message suivant doit apparaître : gcj: no input files

Voici un petit exemple très simple de mise en oeuvre de GCJ.

Exemple du code à compiler :

```
public class Bonjour {
            public static void main(String[] args) {
                        System.out.println("Bonjour");
            }
}
```

Exemple de compilation et d'exécution :

```
D:\java\test\gcj>gcj −o Bonjour Bonjour.java −O −−main=Bonjour
D:\java\test\gcj>dir
 Le volume dans le lecteur D s'appelle DATA
 Le numéro de série du volume est 34B2−159D
 Répertoire de D:\java\test\gcj
01/12/2003  15:19       <DIR>          .
01/12/2003  15:19       <DIR>          ..
01/12/2003  15:19             2 747 919 Bonjour.exe
01/12/2003  15:17                   108 Bonjour.java
01/12/2003  14:07                   141 Bonjour.java.bak
               4 fichier(s)        5 496 087 octets
               2 Rép(s)     560 402 432 octets libres
D:\java\test\gcj>bonjour
Bonjour
D:\java\test\gcj>
```

L'option –o permet de préciser le nom du fichier final généré.

L'option –main= permet de préciser la classe qui contient la méthode main() à lancer par l'exécutable.

GCJ peut être utilisé pour compiler le code source en byte−code grâce à l'option –C.

Exemple :

```
D:\java\test\gcj>gcj −C Bonjour.java
D:\java\test\gcj>dir
 Le volume dans le lecteur D s'appelle DATA
 Le numéro de série du volume est 34B2−159D
 Répertoire de D:\java\test\gcj
01/12/2003  15:29       <DIR>          .
01/12/2003  15:29       <DIR>          ..
01/12/2003  15:29                   389 Bonjour.class
01/12/2003  15:19             2 747 919 Bonjour.exe
01/12/2003  15:17                   108 Bonjour.java
               4 fichier(s)        2 748 557 octets
               2 Rép(s)     563 146 752 octets libres
```

Le byte généré est légèrement plus compact que celui généré par la commande javac du jdk 1.4.1

Exemple :

```
D:\java\test\gcj>javac Bonjour.java
D:\java\test\gcj>dir
 Le volume dans le lecteur D s'appelle DATA
 Le numéro de série du volume est 34B2-159D
 Répertoire de D:\java\test\gcj
01/12/2003  15:29       <DIR>          .
01/12/2003  15:29       <DIR>          ..
01/12/2003  15:31                   405 Bonjour.class
01/12/2003  15:19            2 747 919 Bonjour.exe
01/12/2003  15:17                   108 Bonjour.java
               4 fichier(s)        2 748 573 octets
               2 Rép(s)     563 146 752 octets libres
```

L'option –d permet de préciser un répertoire qui va contenir les fichiers .class généré par l'option –C.

46.5.3. Argo UML

Argo UML est un projet open source écrit en java qui vise à developper un outil de modélisation UML 1.1. Il est possible de créer des diagrammes UML et de générer le code Java correspondant au diagrammes de classes. Une option permet de créer les diagrammes de classes à partir du code source java.

http://argouml.tigris.org/

Cet outil n'est pas encore en version finale mais la version 0.9.5 offre de nombreuses fonctionnalités.

46.5.4. Poseidon UML

http://www.gentleware.com/products/index.php3

46.5.5. Artistic Style

Artistic Style est un outil open source qui permet d'indenter et de formatter un code source C, C++ et java

http://astyle.sourceforge.net

Cet outil possèdent de nombreuses options de formattage de fichiers source. Les options les plus courantes pour un code source java sont :

astyle -jp --style=java nomDuFichier.java

Par défaut, l'outil conserve le fichier original en le suffixant par .orig.

46.5.6. Ant

Ant est un outil du projet jakarta pour réaliser la compilation de projet java. C'est un équivalent à l'outil make sous Unix mais il est écrit en java et donc indépendant de toute plate−forme. Il permet donc la recompilation du projet sur toute plate−forme équipée d'un JVM.

http://jakarta.apache.org/ant

Ant utilise un fichier de paramètres (buildfile) pour la compilation du projet au format XML.

46.5.7. Castor

Castor est un framework open source pour le mapping entre des données et des objets relationnels.

http://castor.exolab.org/

46.5.8. Beanshell

Beanshell est un interpréteur de scripts qu'il est possible d'intégrer dans une application.

http://www.beanshell.org/

46.5.9. Junit

JUnit JUnit est un framework open source pour réaliser et automatiser des tests unitaires et des tests de non régression.

Les cas de tests doivent être implémentés dans des méthodes d'une classe dédiée qui hérite de la classe TestCase.

Il est possible de rassembler plusieurs cas de tests dans une suite de tests qui est une classe qui hérite de la classe TestSuite.

Les tests sont exécutés par un objet de type TestRunner sur la console, dans une application AWT ou dans une application Swing.

http://www.junit.org/

Un chapitre particulier de ce didacticiel est dédié à l'utilisation de JUnit

46.6. Les MOM

Les Middleware Oriented Message sont des outils qui permettent l'échange de messages entre des composants d'une application ou entre applications. Pour pouvoir les utiliser avec Java, ils doivent implémenter l'API JMS de Sun.

46.6.1. OpenJMS

OpenJMS est une implémentation open source des spécifications JMS 1.0.2.

http://openjms.sourceforge.net/

46.6.2. Joram

Joram est l'acronyme de Java Open Reliable Asynchronous Messaging. c'est une implémentation open source des spécifications JMS 1.1.

http://www.objectweb.org/joram/

46.6.3. OSMQ

Open Source Message Queue (OSMQ) est un middleware orienté message développé en open source par Boston System Group.

http://www.osmq.org/

47. JavaDoc

Javadoc est un outil fourni par Sun avec le JDK pour permettre la génération d'une documentation technique à partir du code source.

Ce chapitre contient plusieurs sections :

47.1. La documentation générée

Pour générer la documentation, il faut invoquer l'outil javadoc. Javadoc recrée à chaque utilisation la totalité de la documentation.

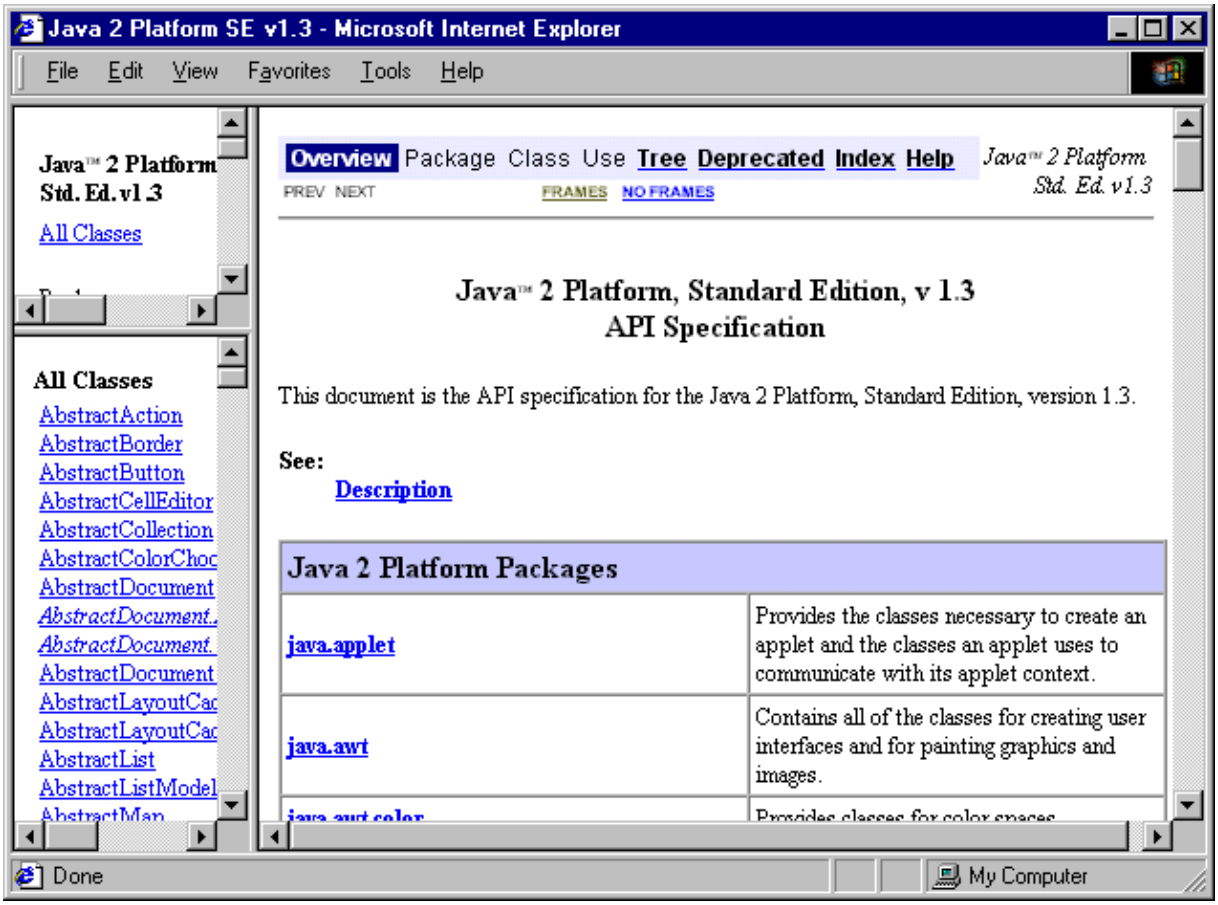
La documentation générée est par défaut au format HTML.

Pour formatter la documentation, javadoc utilise une doclet. Une doclet permet de préciser le format de la documentation générée. Par défaut, javadoc propose un doclet qui génère une documentation au format HTML. Il est possible de définir sa propre doclet pour changer le contenu ou le format de la documentation (pour par exemple, générer du RTF ou du XML).

Par défaut , la documentation générée contient les éléments suivants :

- un fichier html par classe ou interface qui contient le détail chaque élément de la classe ou interface
- un fichier html par package qui contient un résumé du contenu du package
- un fichier overview−summary.html
- un fichier overview−tree.html
- un fichier deprecated−list.html
- un fichier serialized−form.html
- un fichier overview−frame.html
- un fichier all−classe.html
- un fichier package−summary.html pour chaque package
- un fichier package−frame.html pour chaque package
- un fichier package−tree.html pour chaque package

La documentation de l'API java fourni par Sun est réalisée grâce à javadoc :



47.2. Les commentaires de documentation

Javadoc s'appuie sur le code source et sur un type de commentaires particuliers pour obtenir des données supplémentaires aux éléments qui composent le code source. Ces commentaires suivent des règles précises.

Ces commentaires commencent par /** et finissent par */ et contiennent toujours au minimum une phrase qui est un résumé de le l'élément. Si le commentaire utilise plusieurs lignes, javadoc ignore les premiers caractères d'espacement ainsi que le premier caractères * qui suit ces caractères. Ceci permet d'utiliser le caractère * pour aligner le contenu du commentaires

Il est possible de faire suivre cette suivre cette phrase d'un texte descriptif plus complet.

Il est possible d'utiliser des tags HTML pour formatter le texte : il ne faut pas utiliser de tags HTML de structure tel que Hn, HR ... qui sont utilisés par javadoc pour formatter la documentation.

Enfin il est possible d'utiliser des tags prédéfinis par javadoc pour fournir des informations plus précises sur des composants particuliers de l'élément (auteur, parametres, valeur de retour). Ces tags sont définis pour un ou plusieurs type d'élément.

Exemple (code Java 1.0) :

```
/**
 * un commentaire javadoc
 */
```

Par défaut, javadoc prend en compte les éléments suivants : les classes, les interfaces, les méthodes et les champs public et protected. Le placement du commentaire de documentation dans le code source est important. Le documentaire est associé à l'élément qui suit immédiatement le commentaire. Il faut ainsi pour faire précéder l'élément concerné par sa documentation et ne déclarer qu'une seule entité par ligne pour pouvoir lui associer la documentation.

47.3. Les tags définis par javadoc

Javadoc défini plusieurs tags qui permettent de précises certains composants de l'élément décrit de façon standardisée. Ces tags commencent tous par le caractère arobase @. Il existe deux types de tags :

- les tags standards : leur syntaxe est la suivante @tag
- les tags qui seront remplacés par une valeur : la syntaxe est la suivante {@tag}

Pour pouvoir être interprétés les tags standards doivent obligatoirement commencer en début de ligne.

Tag	Rôle	élément concerné	version du JDK
@author	permet de préciser l'auteur de l'élement	classe et interface	1.0
@deprecated	permet de préciser qu'un élément est déprécié	package, classe, interface, méthode et champ	1.1
{@docRoot}	représente le chemin relatif du répertoire principal de génération de la documentation		1.3
@exception	permet de préciser une exception qui peut être levée par l'élément	méthode	1.0
{@link}	permet d'inserer un lien vers un élément de la documentation dans n'importe quel texte	package, classe, interface, methode, champ	1.2
@param	permet de préciser un paramètre de l'élément	constructeur et méthode	1.0
@see	permet de préciser un élément en relation avec l'élément documenté	package, classe, interface, champ	1.0
@serial		classe, interface	1.2
@serialData		methode	1.2
@serialField		classe, interface,	1.2
@since	permet de préciser depuis quelle version l'élément aété ajouté	package, classe, interface, méthode et champ	1.1
@throws	identique à @exception	méthode	1.2
@version	permet de préciser le numéro de version de l'élément	classe et interface	1.0
@return	permet de préciser la valeur de retour d'un élément	méthode	1.0

47.3.1. Le tag @author

Ce tag permet de préciser le nom du ou des auteurs du code. Ce tag doit être utilisé uniquement pour un élement de type classe ou interface.

La syntaxe est la suivante :

@author nom_de_l_auteur

Ce tag génère une entrée Author: avec le nom de l'auteur dans la documentation. Par défaut, ce tag n'est pas pris en compte par javadoc. Pour qu'il soit pris en compte il faut utiliser l'option −author.

Pour préciser plusieurs noms, il faut les séparer par une virgule ou utiliser plusieurs tags chacun contenant un nom.

47.3.2. Le tag @deprecated

Ce tag permet de donner des précisions sur un élément déprécié (deprecated).

Ce tag doit être utilisé uniquement pour un élement de type classe ou interface.

La syntaxe est la suivante :

@deprecated explication

Il est utile de préciser depuis quelle version l'élément est déprécié et de préciser si un autre élément le remplace.

Ce tag génère une entrée Deprecated avec l'explication dans la documentation.

Ce tag est particulier car il est le seul reconnu par le compilateur : celui ci prend note de cet attribut lors de la compilation pour permettre d'informer les utilisateurs de cet élément.

47.3.3. La tag @exception

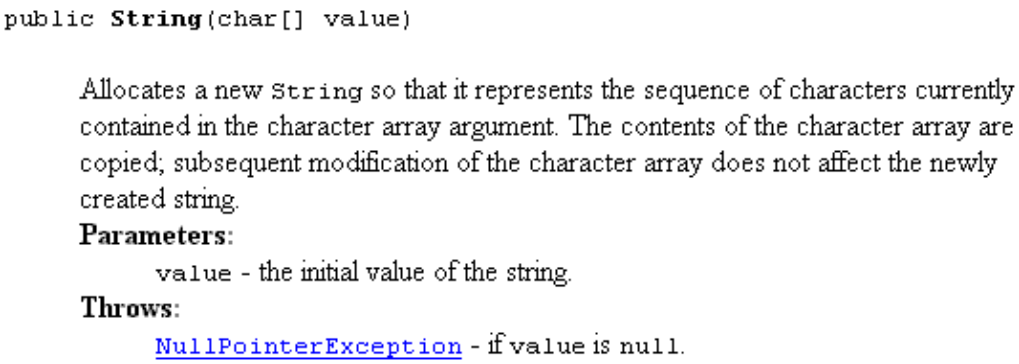
Ce tag permet de fournir des informations sur une exception qui peut être levée. Il faut le faire suivre du nom complétement qualifié de l'exception et d'une description des conditions de sa levée.

Ce tag doit être utilisé uniquement pour un élement de type méthode.

La syntaxe est la suivante :

@exception nom_de_la_classe description_de_l_exception

Exemple extrait de la documentation de l'API du JDK :



Il faut utiliser un tag @exception pour chaque exception déclarée dans la signature de la méthode.

47.3.4. Le tag @param

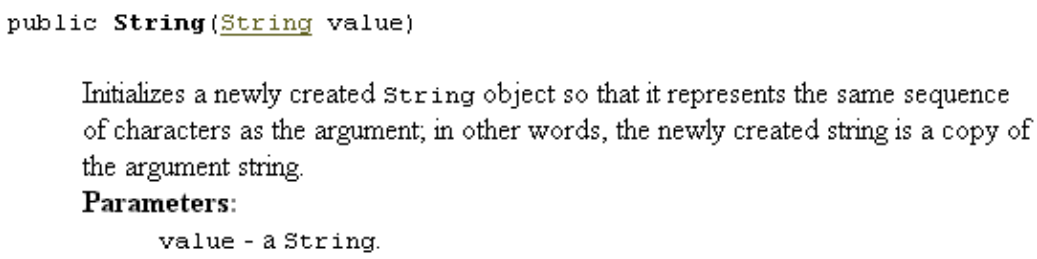
Ce tag permet de fournir des informations sur les paramètres. Ce tag doit être utilisé uniquement pour un élement de type constructeur ou méthode.

La syntaxe est la suivante :

@param nom_du_parametre description_du_parametre

Ce tag génère une ligne dans la section Parameters: avec son nom et description dans la documentation. La description peut tenir sur plusieurs lignes.

Exemple extrait de la documentation de l'API du JDK :



Par défaut, ce tag n'est pas pris en compte par javadoc. Pour qu'il soit pris en compte il faut utiliser l'option −author.

Il faut utiliser un tag @param pour chaque paramètre en respectant l'ordre des paramètres dans la signature.

47.3.5. Le tag @return

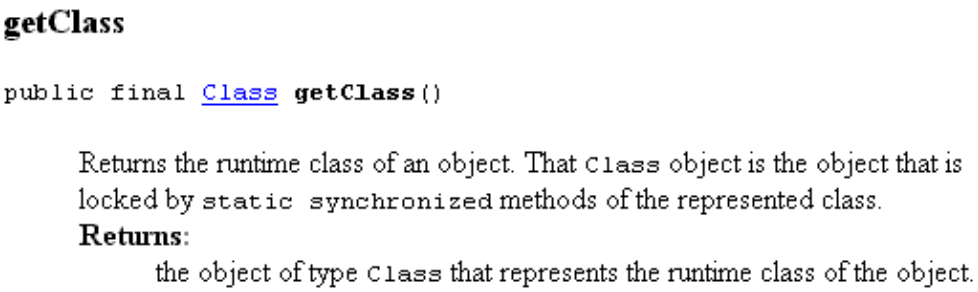
Ce tag permet de préciser la valeur de retour. Ce tag doit être utilisé uniquement pour un élement de type méthode qui renvoie une valeur.

La syntaxe est la suivante :

@return description_retour

Ce tag génère une ligne dans la section Returns: avec sa description dans la documentation. La description peut tenir sur plusieurs lignes.

Exemple extrait de la documentation de l'API du JDK :



47.3.6. La tag @see

Ce tag permet de définir des liens vers d'autres éléments de l'API.

```
public static String valueOf(Object obj)
```

Returns the string representation of the `Object` argument.

Parameters:

`obj` - an `Object`.

Returns:

if the argument is `null`, then a string equal to `"null"`; otherwise, the value of `obj.toString()` is returned.

See Also:

`Object.toString()`

47.3.7. Le tag @since

Ce tag permet de preciser depuis quelle version l'élément est utilisable. Ce tag peut être utilisé avec tous les élements.

La syntaxe est la suivante :

@since numero_de_version

Ce tag génère une ligne dans la section Since: avec son numéro de version.

Exemple extrait de la documentation de l'API du JDK :

```
public byte[] getBytes()
```

Convert this `String` into bytes according to the platform's default character encoding, storing the result into a new byte array.

Returns:

the resultant byte array.

Since:

JDK1.1

47.3.8. Le tag @throws

Ce tag est équivalent au tag @exception

47.3.9. Le tag @version

Ce tag permet de préciser la version d'un élément. Ce tag doit être utilisé uniquement pour un élement de type classe ou interface.

La syntaxe est la suivante :

@version description_de_la_version

Ce tag génère une entrée Version: avec la description de la version dans la documentation. Par défaut, ce tag n'est pas pris en compte par javadoc. Pour qu'il soit pris en compte il faut utiliser l'option −version.

47.4. Exemples

Exemple :

```
/**
  * Commentaire sur le role de la methode
  * @param val la valeur a traiter
  * @since 1.0
  * @return Rien
  * @deprecated Utiliser la nouvelle methode XXX
  */
   public void maMethode(int val) {

   }
```

Résultat :

---

maMethode

```
public void maMethode(int val)
```

Deprecated. *Utiliser la nouvelle methode XXX*

Commentaire sur le role de la methode

Parameters:
`val` - la valeur a traiter
Returns:
Rien
Since:
1.0

47.5. Les fichiers pour enrichir la documentation des packages

Javadoc permet de fournir un moyen de documenter les packages car ceux ci ne disposent de code source particulier : définir des fichiers dont le nom est particulier.

Ces fichiers doivent être placés dans le répertoire désigné par le package.

Le fichier package.html contient une description du package au format HTML. En plus, il est possible d'utiliser les tags @deprecated, @link, @see et @since.

Le fichier overview.html permet de fournir un résumé de plusieurs packages au format html. Ce fichier doit être placé dans le répertoire qui inclus les packages décrits.

La suite de ce chapitre sera développée dans une version future de ce document

48. Java et UML

La suite de ce chapitre sera développée dans une version future de ce document

Le but d'UML est de modéliser un système en utilisant des objets. L'orientation objet de Java ne peut qu'inciter à l'utiliser avec UML. La modélisation proposée par UML repose sur 9 diagrammes.

Ce chapitre contient plusieurs sections :

48.1. Présentation de UML

UML qui est l'acronyme d'Unified Modeling Language est aujourd'hui indissociable de la conception objet. UML est le résultat de la fusion de plusieurs méthodes de conception objet des pères d'UML étaient les auteurs : Jim Rumbaugh (OMT), Grady Booch (Booch method) et Ivar Jacobson (use case).

UML a adopté et normalisé par l'OMG (Object Management Group) en 1997.

D'une façon général, UML est une représentation standardisée d'un système orienté objet.

UML n'est pas une méthode de conception mais notation graphique normalisée de présentation de certains concepts pour modéliser des systèmes objets. En particulier, UML ne précise pas dans quel ordre et comment concevoir les différents diagrammes qu'il défini. Cependant, UML est indépendant de toute méthode de conception et peut être utilisé avec n'importe lequel de ces processus.

Un standard de présentation des concepts permet de faciliter le dialogue entre les différents autres acteurs du projet : les autres analystes, les développeurs, et même les utilisateurs.

UML est composé de neuf diagrammes :

- des cas d'utilisation
- de séquence
- de collaboration
- d'états−transitions
- d'activité
- de classes
- d'objets
- de composants
- de deploiement

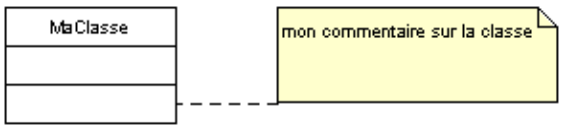
UML regroupe ces neufs diagrammes dans trois familles :

- les diagrammes statiques (diagrammes de classes, d'objet et de cas d'utilisation)
- les diagrammes dynamiques (diagrammes d'activité, de collaboration, de séquence, d'état−transitions et de cas d'utilisation)
- les diagrammes d'architecture : (diagrammes de composants et de déploiements)

48.2. Les commentaires

Utilisable dans chaque diagramme, UML propose une notation particulière pour indiquer des commentaires.

Exemple :



48.3. Les cas d'utilisation (uses cases)

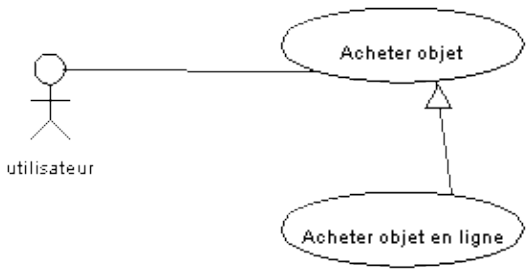
Ils sont développés par Ivar Jacobson et permettent de modéliser des processus métiers en les découpant en cas d'utilisation.

Ce diagramme permet de représenter les fonctionnalités d'un système.Il se compose :

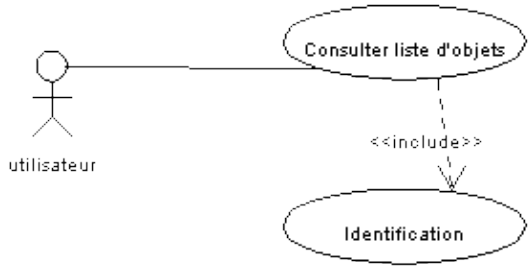
- d'acteurs : ce sont des entités qui utilisent le système à représenter
- les cas d'utilisation : ce sont des fonctionnalités proposées par le système

Un acteur n'est pas une personne désignée : c'est une entité qui joue un rôle dans le système. Il existe plusieurs types de relations qui associent un acteur et un cas d'utilisation :

- la généralisation : cette relation peut être vue comme une relation d'héritage. Un cas d'utilisation enrichie un autre cas en le spécialisant



- l'extension (stéréotype <<extend>>) : le cas d'utilisation complète un autre cas d'utilisation
- l'inclusion (stéréotype <<include>>) : le cas d'utilisation utilise un autre cas d'utilisation



Les cas d'utilisation sont particulièrement intéressant pour recenser les différents acteurs et les différentes fonctionnalités d'un système.

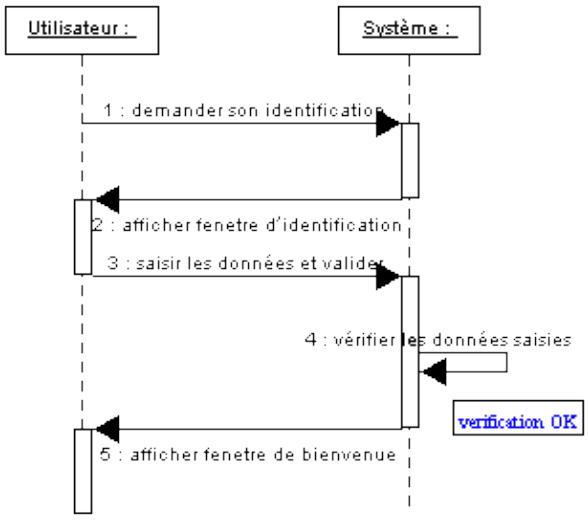
La simplicité de ce diagramme lui permet d'être rapidement compris par des utilisateurs non informaticien. Il est d'ailleurs très important de faire participer les utilisateurs tout au long de son évolution.

Le cas d'utilisation est ensuite détaillé en un ou plusieurs scénarios. Un scénario est une suite d'échanges entre des acteurs et le système pour décrire un cas d'utilisation dans un contexte particulier. C'est est un enchainement précis et ordonné d'opérations pour réaliser le cas d'utilisation.

Si le scénario est trop "volumineux", il peut être judicieux de découper le cas d'utilisation en plusieurs cas d'utilisation et d'utiliser les relations appropriées.

Un scénario peut être représenter par un diagramme de séquence ou sous une forme textuelle. La première forme est très visuelle

Exemple :



La seconde facilite la représentation des opérations alternatives.

Les cas d'utilisation permettent de modéliser des concepts fonctionnels. Il ne précise pas comment chaque opération sera implémentée techniquement. Il faut rester le plus abstrait possible dans les concepts qui s'approchent de la partie technique.

Le découpage d'un système en cas d'utilisation n'est pas facile car il faut trouver un juste mileu entre un découpage faible (les scénarios sont importants) et un découpage faible (les cas d'utilisation se réduisent à une seule opération).

48.4. Le diagramme de séquence

Il permet de modéliser les échanges de message entre les différents objets dans le contexte d'un scénario précis

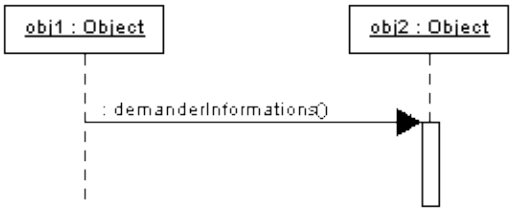
Il permet de répresenter les intéractions entre differentes entités. Il s'utilise essentiellement pour décrire les scénarios d'un cas d'utilisation (les entités sont les acteurs et le système) ou décrirer des échanges entre objets.

Dans le premier cas, les intéractions sont des actions qui sont réalisées par une entité.

Dans le sécond cas, les itérations sont des appels de méthode.

Les itérations peuvent être de deux types :

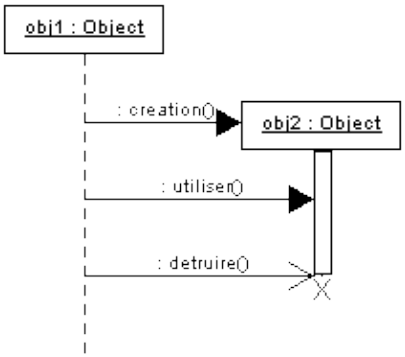
- synchrone : l'émetteur attend une réponse du récepteur



- asynchrone : l'emetteur poursuit son execution sans attendre de réponse

Un diagramme de séquence peut aussi représenter le cycle de vie d'un objet.

Exemple :



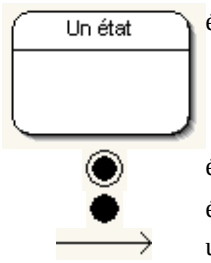
48.5. Le diagramme de collaboration

Il permet de modéliser la collaboration entre les différents objets.

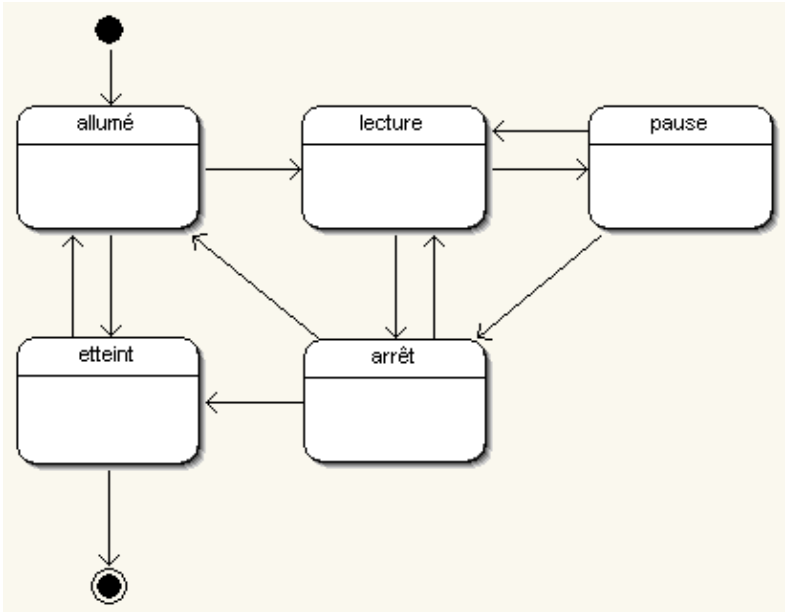
48.6. Le diagramme d'états−transitions

Un diagramme d'état permet de modéliser les différents états d'une entité, en générale une classe. L'ensemble de ces états est connu.

Ce diagramme se compose de plusieurs éléments principaux :

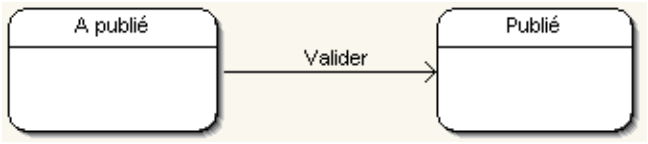


Exemple :



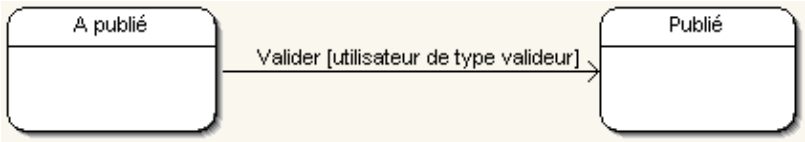
Les transitions sont des événements qui permettent de passer d'un état à un autre : chaque transition possède un sens qui précise l'état de départ et l'état d'arrivée (du côté de la flèche). Une transition peut avoir un nom qui permet de la préciser.

Exemple :



Il est possible d'ajouter une condition à une transition qui est expression booléenne qui sera vérifiée lors d'une demande de transition. Cette condition est indiquée entre crochet.

Exemple :



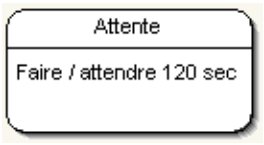
Dans un état, il est possible de préciser des actions ou des activités qui sont des traitements à réaliser dans un état. Celles ci sont décrites avec une étiquette qui désigne le moment de l'exécution.

Une action est un traitement cours. Une activité est un traitement durant tout ou partie de la durée de maintient de l'état.

Plusieurs étiquettes standards sont définies :

- entrée (entry) : action réalisé à l'entrée dans l'état
- sortie (exit) : action réalisée à la sortie de l'état
- faire (do) : activité exécuté durant l'état

Exemple :



Il est aussi possible de définir des actions internes

48.7. Le diagramme d'activités

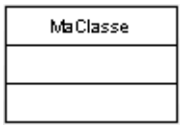
48.8. Le diagramme de classes

Ce schéma représente les différentes classes : il détaille le contenu de chaque classe mais aussi les relations qui peuvent exister entre les différentes classes.

Une classe est représentée par un rectangle séparée en trois parties :

- la première partie contient le nom de la classe
- la seconde contient les attributs de la classe
- la dernière contient les méthodes de la classe

Exemple :



Exemple :

```
public class MaClasse {
}
```

Les attributs d'une classe

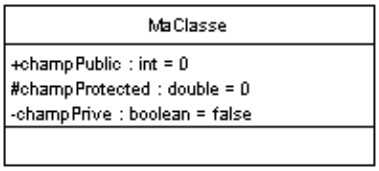
Pour définir un attribut, il faut préciser son nom suivi du caractère ":" et du type de l'attribut.

Le modificateur d'accès de l'attribut doit précéder son nom et peut prendre les valeurs suivantes :

Caractère	Rôle
+	accès public
#	accès protected
–	accès private

Une valeur d'initialisation peut être précisée juste après le type en utilisant le signe "=" suivi de la valeur.

Exemple :



Exemple :

```
public class MaClasse {

  public int champPublic = 0;
  protected double champProtected = 0;
  private boolean champPrive = false;
}
```

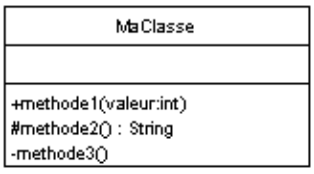
Les méthodes d'une classe

Les modificateurs sont identiques à ceux des attributs.

Les paramètres de la méthodes peuvent être précisés en les indiquant entre les parathèses sous la forme nom : type.

Si la méthode renvoie une valeur son type doit être précisé après un signe ":".

Exemple :



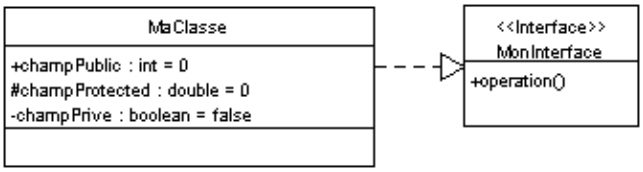
Exemple :

```
public class MaClasse {

  public void methode1(int valeur){
  }

  protected String methode2(){
  }

  private void methode3(){
  }
}
```

Il n'est pas obligatoire d'inclure dans le diagramme tous les attributs et toutes les méthodes d'une classe : seules les entités les plus significatives et utiles peuvent être mentionnées.

L'implémentation d'une interface

Exemple :

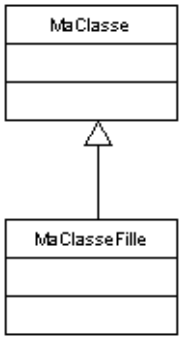


Exemple :

```
public class MaClasse implements MonInterface {
    public int champPublic = 0;
    protected double champProtected = 0;
    private boolean champPrive = false;

    public operation() {

    }
}
```

La relation d'héritage

Exemple :



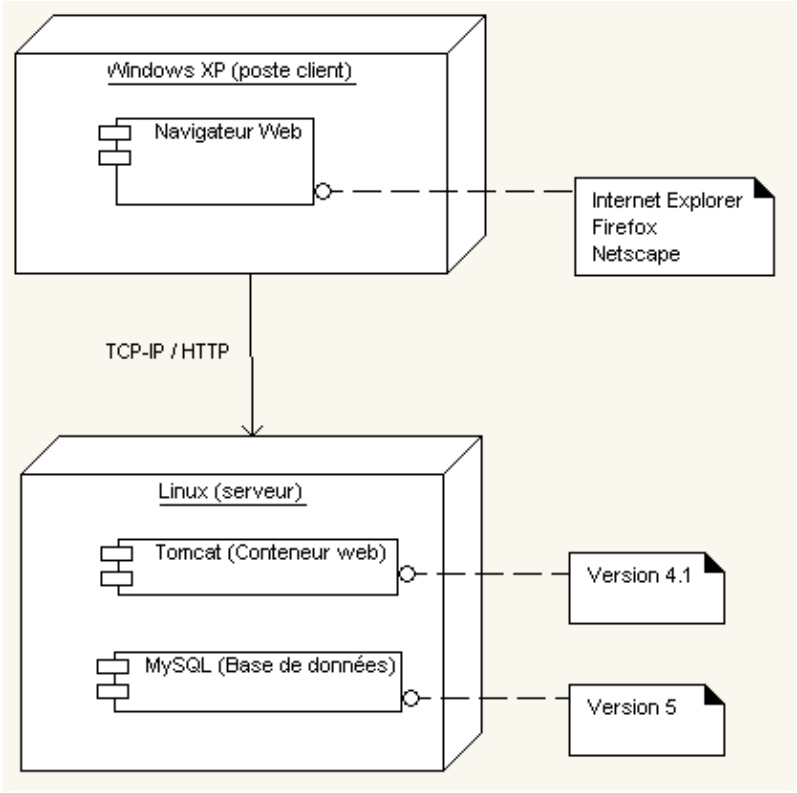
Exemple :

```
public class MaClasseFille extends MaClasse {

}
```

48.9. Le diagramme d'objets

48.10. Le diagramme de composants

48.11. Le diagramme de déploiement



49. Des normes de développement

Le but de ce chapitre est de proposer un ensemble de conventions et de règles pour faciliter la compréhension et donc la maintenance du code.

Ces règles ne sont pas à suivre explicitement à la lettre : elles sont uniquement présentées pour inciter le ou les développeurs à définir et à utiliser des règles dans la réalisation de son code surtout dans le cadre d'un travail en équipe. Les règles proposées sont celles couramment utilisées. Il n'existe cependant pas de règles absolues et chacun pourra utiliser tout ou partie des règles proposées.

La définition de conventions et de règles est importante pour plusieurs raisons :

- La majorité du temps passé à coder est consacré à la maintenante évolutive et corrective d'une application
- Ce n'est pas toujours voir rarement l'auteur du code qui effectue ces maintenance
- ces règles facilitent la lisibilité et donc la compréhension du code

Le contenu de ce document est largement inspiré par les conventions de codage proposées par Sun à l'URL suivante : http://java.sun.com/docs/codeconv/index.html

Ce chapitre contient plusieurs sections :

49.1. Les fichiers

Java utilise des fichiers pour stocker les sources et le byte code des classes.

49.1.1. Les packages

Les packages permettent de grouper les classes sous une forme hiérarchisée. Le choix des critères de regroupement est laissé aux développeurs.

Il est préférable de regrouper les classes par packages selon des critères fonctionnels.

Les fichiers inclus dans un package doivent être insérer dans une arborescence de répertoires équivalentes.

49.1.2. Le nom de fichiers

Chaque fichier source ne doit contenir qu'une seule classe ou interface publique. Le nom du fichier doit être identique au nom de cette classe ou interface publique en respectant la casse.

Il faut éviter dans ce nom d'utiliser des caractères accentués qui ne sont pas toujours utilisables par tous les systèmes d'exploitation.

Les fichiers sources ont pour extension .java car le compilateur javac fourni avec le J.D.K. utilise cette extension

Exemple :

```
javac MaClasse.java
```

Les fichiers binaires contenant le byte−code ont pour extension .class car le compilateur génère un fichier avec cette extension à partir du fichier source .java correpondant. De plus, elle est obligatoire pour l'interpréteur Java qui l'ajoute automatiquement au nom du fichier fourni en paramètre.

Exemple :

```
java MaClasse
```

49.1.3. Le contenu des fichiers sources

Un fichier ne devrait pas contenir plus de 2 000 lignes de code.

Des interfaces ou classes privées ayant une relation avec la classe publique peuvent être rassemblées dans un même fichier. Dans ce cas, la classe publique doit être la première dans le fichier.

Chaque fichier source devrait contenir dans l'ordre

1. un commentaire concernant le fichier
2. les clauses concernant la gestion des packages (la déclaration et les importations)
3. les déclarations de classes ou de l'interface

49.1.4. Les commentaires de début de fichier

Chaque fichier source devrait commencer par un commentaire multi−lignes contenant au minimum des informations sur le nom de la classe, la version, la date, éventuellement le copyright et tout autres commentaires utiles :

Exemple :

```
/*
 * Nom de classe : MaClasse
 *
 * Description   : description de la classe et de son rôle
 *
 * Version       : 1.0
 *
 * Date          : 23/02/2001
 *
 * Copyright     : moi
 */
```

49.1.5. Les clauses concernant les packages.

La première ligne de code du fichier devrait être une clause package indiquant à quel paquetage appartient la classe. Le fichier source doit obligatoirement être inclus dans une arborescence correspondante au nom du package.

Il faut indiquer ensuite l'ensemble des paquetages à importer : ceux dont les classes vont être utilisées dans le code.

Exemple :

```
package monpackage;

import java.util.*;
import java.text.*;
```

49.1.6. La déclaration des classes et des interfaces

Les différents éléments qui composent la définition de la classe ou de l'interface devraient être indiqués dans l'ordre suivant :

1. les commentaires au format javadoc de la classe ou de l'interface
2. la déclaration de la classe ou de l'interface
3. les variables de classes (déclarées avec le mot clé static) triées par ordre d'accessibilité : d'abord les variables déclarées public, protected, package friendly (sans modificateur d'accès) et enfin private
4. les variables d'instances triées par ordre d'accessibilité : d'abord les variables déclarées public, protected, package friendly (sans modificateur d'accès) et enfin private
5. le ou les constructeurs
6. les méthodes : elles seront regroupées par fonctionnalités plutôt que selon leur accessibilité

49.2. La documentation du code

Il existe deux types de commentaires en java :

- les commentaires de documentation : ils permettent en respectant quelques règles d'utiliser l'outil javadoc fourni avec le J.D.K. qui formate une documentation des classes, indépendante de l'implémentation du code,
- les commentaires de traitements : ils fournissent un complément d'information dans le code lui−même.

Les commentaires ne doivent pas être entourés par de grands cadres dessinés avec des étoiles ou d'autres caractères.

Les commentaires ne devraient pas contenir de caractères spéciaux tels que le saut de page.

49.2.1. Les commentaires de documentation

Les commentaires de documentation utilisent une syntaxe particulière utilisée par l'outil javadoc de Sun pour produire une documentation standardisée des classes et interfaces au format HTML. La documentation de l'API du J.D.K. est le résultat de l'utilisation de cet outil de documentation

49.2.1.1. L'utilisation des commentaires de documentation

Cette documentation concerne les classes, les interfaces, les constructeurs, les méthodes et les champs.

La documentation est définie entre les caractères /** et */ selon le format suivant :

Exemple :

```
/**
 *  Description de la methode
 */
public void maMethode() {
```

La première ligne de commentaires ne doit contenir que /**

Les lignes de commentaires suivantes doivent obligatoirement commencer par un espace et une étoile. Toutes les premières étoiles doivent être alignées.

La dernière ligne de commentaires ne doit contenir que */ précédé d'un espace.

Un tel commentaire doit être défini pour chaque entité : une classe, une interface et chaque membre (variables et méthodes).

Javadoc définit un certain nombre de tags qu'il est possible d'utiliser pour apporter des précisions sur plusieurs informations.

Ces tags permettent de définir des caractéristiques normalisées. Il est possible d'inclure dans les commentaires des tags HTML de mise en forme (PRE, TT, EM ...) mais il n'est pas recommandé d'utiliser des tags HTML de structure tel que Hn, HR, TABLE ... qui sont utilisés par javadoc pour formater la documentation

Il faut obligatoirement faire précéder l'entité documentée par son commentaire car l'outil associe la documentation à la déclaration de l'entité qui la suit.

49.2.1.2. Les commentaires pour une classe ou une interface

Pour les classes ou interfaces, javadoc définit les tags suivants : @see, @version, @author, @copyright, @security, @date, @revision, @note

Les tags @copyright, @security, @date, @revision et @note ne sont pas traités par javadoc.

Exemple :

```
/**
 * NomClasse – description de la classe
 * explication supplémentaire si nécessaire
 *
 * @version1.0
 *
 * @see UneAutreClasse
 * @author Jean Michel D.
 * @copyright (C) moi 2001
 * @date 01/09/2000
 * @notes notes particulières sur la classe
 *
 * @revision référence
 *           date 15/11/2000
 *           author Michel M.
 *           raison  description
 *           description supplémentaire
 */
```

49.2.1.3. Les commentaires pour une variable de classe ou d'instance

49.2.1.4. Les commentaires pour une méthode

Pour les méthodes, javadoc définit les tags suivants : @see, @param, @return, @exception, @author, @note

Le tag @note n'est pas traité par javadoc.

Exemple :

```
/**
 * nomMethode – description de la méthode
```

```
 *              explication supplémentaire si nécessaire
 *
 *     exemple d'appel de la methode
 * @return      description de la valeur de retour
 * @param       arg1 description du 1er argument
 *    :           :         :
 * @param       argN description du Neme argument
 * @exception   Exception1  description de la première exception
 *    :           :         :
 * @exception ExceptionN  description de la Neme exception
 *
 * @see UneAutreClasse#UneAutreMethode
 * @author   Jean Dupond
 * @date     12/02/2001
 * @note     notes particulières.
 */
```

Remarques :

- @return ne doit pas être utilisé avec les constructeurs et les méthodes sans valeur de retour (void)
- @param ne doit pas être utilisé si il n'y a pas de paramètres
- @exception ne doit pas être utilisé si il n'y pas d'exception propagée par la méthode
- @author doit être omis si il est identique à celui du tag @author de la classe
- @note ne doit pas être utilisé si il n'y a pas de note

49.2.2. Les commentaires de traitements

Ces commentaires doivent ajouter du sens et des précisions au code : ils ne doivent pas reprendre ce que le code exprime mais expliquer clairement son rôle.

Tous les commentaires utiles à une meilleure compréhension du code et non inclus dans les commentaires de documentation seront insérés avec des commentaires de traitements. Il existe plusieurs styles de commentaires :

- les commentaires sur une ligne
- les commentaires sur une portion de ligne
- les commentaires multi−lignes

Il est conseillé de mettre un espace après le délimiteur de début de commentaires et avant le délimiteur de fin de commentaires lorsqu'il y en a un, afin d'améliorer sa lisibilité.

49.2.2.1. Les commentaires sur une ligne

Ces commentaires sont définis entre les caractères /* et */ sur une même ligne

Exemple :

```
if (i < 10) {
   /* commentaires utiles au code */
   ...
}
```

Ce type de commentaires doit être précédé d'une ligne blanche et doit suivre le niveau d'indentation courant.

49.2.2.2. Les commentaires sur une portion de ligne

Ce type de commentaires peut apparaître sur la ligne de code qu'elle commente mais il faut inclure un espace conséquent qui permette de séparer le code et le commentaire.

Exemple :

```
i++;                        /* commentaires utiles au code */
```

Si plusieurs lignes qui se suivent contiennent chacune un tel commentaire, il faut les aligner :

Exemple :

```
i++;                        /* commentaires utiles au code */
j++;                        /* second commentaires utiles au code */
```

49.2.2.3. Les commentaires multi−lignes

Exemple :

```
/*
 * Commentaires utiles au code
 */
```

Ce type de commentaires doit être précédé d'une ligne blanche et doit suivre le niveau d'indentation courant.

49.2.2.4. Les commentaires de fin de ligne

Ce type de commentaire peut délimiter un commentaire sur une ligne complète ou une fin de ligne.

Exemple :

```
i++;         // commentaires utiles au code
```

Ce type de commentaires peut apparaître sur la ligne de code qu'elle commente mais il faut inclure un espace conséquent qui permette de séparer le code et le commentaire.

Si plusieurs lignes qui se suivent contiennent chacune un tel commentaire, il faut les aligner :

Exemple :

```
i++;                        // commentaires utiles au code
j++;                        // second commentaires utiles au code
```

L'usage de cette forme de commentaires est fortement recommandé car il est possible d'inclure celui ci dans un autre de la forme /* */ et ainsi mettre en commentaire un morceau de code incluant déjà des commentaires.

49.3. Les déclarations

49.3.1. La déclaration des variables

Il n'est pas recommandé d'utiliser des caractères accentués dans les identifiants de variables, cela peut éventuellement poser des problèmes dans le cas ou le code est édité sur des systèmes d'exploitation qui ne les gèrent pas correctement.

Il ne doit y avoir qu'une seule déclaration d'entité par ligne.

Exemple :

```
String nom;
String prenom;
```

Cet exemple est préférable à

Exemple :

```
String nom, prenom;          //ce type de déclaration n'est pas recommandée
```

Il faut éviter de déclarer des variables de types différents sur la même ligne même si cela est accepté par le compilateur.

Exemple :

```
int age, notes[];            // ce type de déclaration est à éviter
```

Il est préférable d'aligner le type, l'identifiant de l'objet et les commentaires si plusieurs déclarations se suivent pour retrouver plus facilement les divers éléments.

Exemple :

```
String          nom          //nom de l'eleve
String          prenom       //prenom de l'eleve
int             notes[]      //notes de l'eleve
```

Il est fortement recommandé d'initialiser les variables au moment de leur déclaration.

Il est préférable de rassembler toutes les déclarations d'un bloc au début de ce bloc. (un bloc est un morceau de code entouré par des accolades).

La seule exception concerne la déclaration de la variable utilisée comme index dans une boucle.

Exemple :

```
for (int i = 0 ; i < 9 ; i++) { ... }
```

Il faut proscrire la déclaration d'une variable qui masque une variable définie dans un bloc parent afin de ne pas complexifier inutilement le code.

Exemple :

```
int taille;
...
void maMethode() {
    int taille;
}
```

49.3.2. La déclaration des classes et des méthodes

Il ne doit pas y avoir d'espaces entre le nom d'une méthode et sa parenthèse ouvrante.

L'accolade ouvrante qui définit le début du bloc de code doit être à la fin de la ligne de déclaration.

L'accolade fermante doit être sur une ligne séparée dont le niveau d'indentation correspond à celui de la déclaration.

Une exception tolérée concerne un bloc de code vide : dans ce cas les deux accolades peuvent être sur la même ligne.

La déclaration d'une méthode est précédée d'une ligne blanche.

Exemple :

```
class MaClasse extends MaClasseMere {
   String nom;
   String prenom;
   MaClasse(String nom, String prenom) {
      this.nom = nom;
      this.prenom = prenom;
   }
   void neRienFaire() {}
}
```

Il faut éviter d'écrire des méthodes longues et compliquées : le traitement réalisé par une méthode doit être simple et fonctionnel. Cela permet d'écrire des méthodes réutilisables dans la classe et facilite la maintenance. Cela permet aussi d'éviter la redondance de code.

Java propose deux syntaxes pour déclarer une méthode qui retourne un tableau : la première syntaxe est préférable.

Exemple :

```
public int[] notes() {     // utiliser cette forme
public int notes()[] {
```

Il est fortement recommandé de toujours initialiser les variables locales d'une méthode lors de leur déclaration car contrairement aux variables d'instances, elles ne sont pas implicitement initialisées avec une valeur par défaut selon leur type.

49.3.3. La déclaration des constructeurs

Elle suit les mêmes règles que celles utilisées pour les méthodes.

Il est préférable de définir explicitement le constructeur par défaut (le constructeur sans paramètre). Soit le constructeur par défaut est fourni par le compilateur et dans ce cas il est préférable de le définir soit il existe d'autres constructeurs et dans ce cas le compilateur ne définit pas de constructeur par défaut.

Il est préférable de toujours initialiser les variables d'instance dans un constructeur soit avec les valeurs fournies en paramètres du constructeur soit avec des valeurs par défaut.

Exemple :

```
class Personne {
   String nom;
   String prenom;
   int    age;

   Personne() {
      this( "Inconnu", "inconnu", -1 );
   }
```

```
   Personne( String nom, String prenom, int age ) {
      this.name    = nom;
      this.address = prenom;
      this.age     = age;
   }
}
```

Il est possible d'appeler un constructeur dans un autre constructeur pour faciliter l'écriture.

Il est recommandé de toujours appeler explicitement le constructeur hérité lors de la redéfinition d'un constructeur dans une classe fille grâce à l'utilisation du mot clé super.

Exemple :

```
class Employe extends Personne {
   int matricule;
   Employee() {
      super();
      matricule = -1;
   }

   Employee(String nom, String prenom, int age, int matricule) {
      super(nom, prenom, age);
      this.matricule = matricule;
   }
}
```

Il est conseillé de ne mettre que du code d'initialisation des variables d'instances dans un constructeur et de mettre les traitements dans des méthodes qui seront appelées après la création de l'objet.

49.3.4. Les conventions de nommage des entités

Les conventions de nommage des entités permettent de rendre les programmes plus lisibles et plus faciles à comprendre. Ces conventions permettent notamment de déterminer rapidement quelle entité désigne un identifiant, une classe ou une méthode.

Entités	Règles	Exemple
Les packages	Toujours écrits tout en minuscules (norme java 1.2)	com.entreprise.projet
Les classes, les interfaces et les constructeurs	La première lettre est en majuscule. Si le nom est composé de plusieurs mots, la première lettre de chaque mot doit être en majuscule, ne pas mettre de caractère underscore '_' Le nom d'une classe peut finir par impl pour la distinguer d'une interface qu'elle implémente. Les classes qui définissent des exceptions doivent finir par Exception.	MaClasse MonInterface MaClasse()
Les méthodes	Leur nom devrait contenir un verbe. La première lettre est obligatoirement une minuscule. Si le nom est composé de plusieurs mots, la première lettre de chaque mot doit être en majuscule sans mettre de caractère underscore '_' Les méthodes pour obtenir la valeur d'un champ doivent commencer par get suivi du nom du champ.	public float calculerMontant() {

	Les méthodes pour mettre à jour la valeur d'un champ doivent commencer ser set suivi du nom du champ Les méthodes pour créer des objets (factory) devraient commencer par new ou create Les méthodes de conversion devraient commencer par to suivi par le nom de la classe renvoyée à la suite de la conversion	
Les variables	La première lettre est obligatoirement une minuscule et ne devrait pas être un caractère dollar‘$' ou underscore ‘_' même si ceux ci sont autorisés. Pour les variables d'instances non publiques, certains recommandent de commencer par un underscore pour éviter la confusion avec le nom d'une variable fournie en paramètre d'une méthode tel que le setter. Si le nom est composé de plusieurs mots, la première lettre de chaque mot doit être en majuscule, ne pas mettre de caractère underscore ‘_'. Les noms de variables composés d'un seul caractère doivent être évités sauf pour des variables provisoires (index d'une boucle). Les noms communs pour ces variables provisoires sont i,j,k,m et n pour les entiers et c,d et e pour les caractères.	String nomPersonne; Date dateDeNaissance; int i;
Les constantes	Toujours en majuscules, chaque mots est séparés par un underscore ‘_'. Ces variables doivent obligatoirement être initialisées lors de leur déclaration.	static final int VAL_MIN = 0; static final int VAL_MAX = 9;

49.4. Les séparateurs

L'usage des séparateurs tels que les retours à la ligne, les lignes blanches, les espaces, etc ... permet de rendre le code moins « dense » et donc plus lisibles.

49.4.1. L'indentation

L'unité d'indentation est constituée de 4 espaces. Il n'est pas recommandé d'utiliser les tabulations pour l'indentation.

Il est préférable d'éviter les lignes contenant plus de 80 caractères.

49.4.2. Les lignes blanches

Elles permettent de définir des sections dans le code pour effectuer des séparations logiques.

Deux lignes blanches devraient toujours séparer deux sections d'un fichier source et les définitions des classes et des interfaces.

Une ligne blanche devrait toujours être utilisée dans les cas suivants :

- avant la déclaration d'une méthode,
- entre les déclarations des variables locales et la première ligne de code,
- avant un commentaire d'une seule ligne,
- avant chaque section logique dans le code d'une méthode.

49.4.3. Les espaces

Un espace vide devrait toujours être utilisé dans les cas suivants :

- entre un mot clé et une parenthèse.

Exemple :
`while (i < 10)`

- après chaque virgule dans une liste d'argument
- tous les opérateurs binaires doivent avoir un blanc qui les précèdent et qui les suivent

Exemple :
`a = (b + c) * d`

- chaque expression dans une boucle for doit être séparée par un espace

Exemple :
`for (int i; i < 10; i++)`

- les conversions de type explicites (cast) doivent être suivies d'un espace

Exemple :
`i = ((int) (valeur + 10));`

Il ne faut pas mettre d'espace entre un nom de méthode et sa parenthèse ouvrante.

Il ne faut pas non plus mettre de blanc avant les opérateurs unaires tel que les opérateurs d'incrément ‘++' et de décrément ‘−−‘.

Exemple :
`i++;`

49.4.4. La coupure de lignes

Il arrive parfois qu'une ligne de code soit très longue (supérieure à 80 caractères).

Dans ce cas, il est recommandé de couper cette ligne en une ou plusieurs en respectant quelques règles :

- couper la ligne après une virgule ou avant un opérateur
- aligner le début de la nouvelle ligne au début de l'expression coupée

Exemple :

```
maMethode(parametre1, parametre2, parametre3,
          parametre4, parametre5);
```

49.5. Les traitements

Même si il est possible de mettre plusieurs traitements sur une ligne, chaque ligne ne devrait contenir qu'un seul traitement

Exemple :

```
i = getSize();
i++;
```

49.5.1. Les instructions composées

Elles correspondent à des instructions qui utilisent des blocs de code.

Les instructions incluses dans ce bloc sont encadrées par des accolades et doivent être indentées.
L'accolade ouvrante doit se situer à la fin de la ligne qui contient l'instruction composée.
L'accolade fermante doit être sur une ligne séparée au même niveau d'indentation que l'instruction composée.

Un bloc de code doit être définit pour chaque traitement même si le traitement ne contient qu'une seule instruction. Cela facilite l'ajout d'instructions et évite des erreurs de programmation.

49.5.2. L'instruction return

Elle ne devrait pas utiliser de parenthèses sauf si celle ci facilite la compréhension

Exemple :

```
return;
 return valeur;
 return (isHomme() ? ‘M' : ‘F');
```

49.5.3. L'instruction if

Elle devrait avoir une des formes suivantes :

```
if (condition) {
    traitements;
 }

 if (condition) {
    traitements;
 } else {
    traitements;
 }

 if (condition) {
    traitements;
 } else if (condition) {
    traitements;
 } else {
    traitements;
 }
```

Même si cette forme est syntaxiquement correcte, il est préférable de ne pas utiliser l'instruction if sans accolades :

Exemple :

```
   if (i == 10) i = 0;  // cette forme ne doit pas être utilisée
```

49.5.4. L'instruction for

Elle devrait avoir la forme suivante :

```
for ( initialisation; condition; mise à jour) {
     traitements;
  }
```

49.5.5. L'instruction while

Elle devrait avoir la forme suivante :

```
while (condition) {
     traitements;
  }
```

Si il n'y a pas de traitements, la forme est la suivante :

while (condition);

49.5.6. L'instruction do−while

Elle devrait avoir la forme suivante :

```
  do {
     traitements;
  } while ( condition);
```

49.5.7. L'instruction switch

Elle devrait avoir la forme suivante :

```
  switch (condition) {
  case ABC:
     traitements;
  case DEF:
     traitements;
  case XYZ:
     traitements;
  default:
     traitements;
  }
```

Il est préférable de terminer les traitements de chaque cas avec une instruction break.

Toutes les instructions switch devrait avoir un cas ‘default' en fin d'instruction.

Même si elle est redondante, une instruction break devrait être incluse en fin des traitements du cas ‘default'.

49.5.8. Les instructions try−catch

Elle devrait avoir la forme suivante :

```
try {
    traitements;
 } catch (Exception1 e1) {
    traitements;
 } catch (Exception2 e2) {
    traitements;
 } finally {
    traitements;
 }
```

49.6. Les règles de programmation

49.6.1. Le respect des règles d'encapsulation

Il ne faut pas déclarer de variables d'instances ou de classes publiques sans raison valable.

Il est préférable de restreindre l'accès à la variable avec un modificateur d'accès protected ou private et de déclarer des méthodes respectant les conventions instaurées dans les javaBeans : getXxx() ou isXxx() pour obtenir la valeur et setXxx() pour mettre à jour la valeur.

La création de méthodes sur des variables private ou protected permet d'assurer une protection lors de l'accès à la variable (déclaration des méthodes d'accès synchronized) et éventuellement un contrôle lors de la mise à jour de la valeur.

49.6.2. Les références aux variables et méthodes de classes.

Il n'est pas recommandé d'utiliser des variables ou des méthodes de classes à partir d'un objet instancié : il ne faut pas utiliser objet.methode() mais classe.methode().

Exemple à ne pas utiliser si afficher() est une méthode de classe :

```
MaClasse maClasse = new MaClasse();
maClasse.afficher();
```

Exemple à utiliser si afficher() est une méthode de classe :

```
MaClasse.afficher();
```

49.6.3. Les constantes

Il est préférable de ne pas utiliser des constantes numériques en dur dans le code mais de déclarer des constantes avec des noms explicites. Une exception concerne les valeur −1, 0 et 1 dans les boucles for.

49.6.4. L'assignement des variables

Il n'est pas recommandé d'assigner la même valeur à plusieurs variables sur la même ligne :

Exemple :

```
i = j = k;                 //cette forme n'est pas recommandée
```

Il ne faut pas utiliser l'opérateur d'assignement imbriqué.

Exemple à proscrire :

```
  valeur = (i = j + k ) + m;
```

Exemple à utiliser :

```
 i = j + k;
  valeur = i + m;
```

Il n'est pas recommandé d'utiliser l'opérateur d'assignation = dans une instruction if ou while afin d'éviter toute confusion.

49.6.5. L'usage des parenthèses

Il est préférable d'utiliser les parenthèses lors de l'usage de plusieurs opérateurs pour éviter des problèmes liés à la priorité des opérateurs.

Exemple :

```
if (i == j && m == n)                // à éviter
 if ( (i == j) && (m == n ))          // à utiliser
```

49.6.6. La valeur de retour

Il est préférable de minimiser le nombre d'instruction return dans un bloc de code.

Exemple à eviter :

```
  if (isValide()){
     return true;
  } else {
    return false;
  }
```

Exemple à utiliser :

```
  return isValide();
```

Exemple à éviter :

```
if (isValide()) {
    return x;
 } else return y;
```

Exemple à utiliser :

```
return (isValide() ? x : y)
```

49.6.7. La codification de la condition dans l'opérateur ternaire ? :

Si la condition dans un opérateur ternaire ? : contient un opérateur binaire, cette condition doit être mise entre parenthèses

Exemple :

```
( i >= 0 ) ? i : −i;
```

49.6.8. La déclaration d'un tableau

Java permet de déclarer les tableaux de deux façons :

Exemple :

```
public int[] tableau = new int[10];
 public int tableau[] = new int[10];
```

L'usage de la première forme est recommandée.

50. Les motifs de conception (design patterns)

La suite de ce chapitre sera développée dans une version future de ce document

Le nombre de développement avec des technologies orientées objets augmentant, l'idée de réutiliser des techniques pour solutionner des problèmes courants à abouti aux recensements d'un certain nombre de modèles connus sous les motifs de conception.

Ces modèles sont définis pour pouvoir être utilisés avec un maximum de langage orienté objet.

Le nombre de ces modèles est en constante augmentation. Le but de ce chapitre n'est pas de tous les recenser mais de présenter les plus utilisés et de fournir un ou des exemples de leur mise en oeuvre avec Java.

Il est habituel de regrouper ces modèles communs dans trois grandes catégories :

- les modèles de création (creational patterns)
- les modèles de structuration (structural patterns)
- les modèles de comportement (behavioral patterns)

Le motif de conception le plus connu est surement le modele MVC (Model View Controller) mis en oeuvre en premier avec SmallTalk.

Ce chapitre contient plusieurs sections :

50.1. Les modèles de création

Dans cette catégorie, il existe 5 modèles principaux :

Nom	Rôle
Fabrique (Factory)	Interface qui laisse des sous classes créer des objets
Fabrique abstraite (abstract Factory)	
Monteur (Builder)	

Prototype (Prototype)	Création d'objet à partir d'un prototype
Singleton (Singleton)	Classe qui ne pourra avoir qu'une seule instance

50.1.1. Fabrique (Factory)

50.1.2. Fabrique abstraite (abstract Factory)

50.1.3. Monteur (Builder)

50.1.4. Prototype (Prototype)

50.1.5. Singleton (Singleton)

Ce modèle permet de définir une classe dont il ne pourra y avoir qu'une seule instance. Le modèle assure aussi l'accès à cette unique instance.

Ce modèle est particulièrement utile pour le développement d'objets de type gestionnaire. En effet ce type d'objet doit être unique car il gère d'autres objets par exemple un gestionnaire de logs.

Pour mettre en oeuvre ce modèle, il faut :

- créer une instance de la classe et la stocker dans une variable privée
- empecher l'utilisation du ou des constructeurs
- fournir une méthode qui renvoie l'instance stockée dans la variable privée

Exemple :

```
public class MonSingleton {

  /** Singleton. */
  private static MonSingleton monSingleton = new MonSingleton();

  /**
   * Constructeur de la classe MonSingleton.
   */
  private MonSingleton() {
    super();
  }

  /**
   * Renvoie le singleton.
   */
  public static MonSingleton get() {
    return monSingleton;
  }
```

```
  public void afficher() {
    System.out.println("Singleton");
  }
}
```

Pour vérifier que l'usage du constructeur est impossible, il suffit de compiler une classe qui tente d'en faire usage.

Exemple :

```
public class TestSingleton1 {
  public static void main() {
    MonSingleton ms = new MonSingleton();
    ms.afficher();
  }
}
```

Résultat :

```
C:\java>javac TestSingleton1.java

TestSingleton1.java:4:
No constructor matching MonSingleton() found in class Mon
Singleton.

                MonSingleton ms = new MonSingleton();
                                  ^
1 error
```

Le compilateur indique une erreur car le constructeur a été déclaré private pour empecher son appel.

Pour pouvoir utiliser l'instance de la classe, il faut appeler la méthode qui renvoie l'instance unique.

Exemple :

```
public class TestSingleton2 {
            public static void main(String[] args) {
                        MonSingleton ms = MonSingleton.get();
                        ms.afficher();
            }
}
```

Résultat :

```
C:\java>javac TestSingleton2.java

C:\java>java TestSingleton2
Singleton

C:\java>
```

Il faut impérativement déclarer le ou les constructeurs par défaut et explicitement déclarer un constructeur par défaut pour empecher le compilateur de l'ajouter.

L'instanciation de l'unique instance peut être réalisée de façon statique ou réalisée à la demande. Dans ce cas, la méthode qui renvoie l'instance doit vérifier qu'elle existe et dans la cas contraire, créer l'instance, la stockée dans la variable privée et la renvoyer.

Pour accentuer encore l'assurance de l'unicité de l'instance, il peut être utile de déclarer la classe finale pour éviter que celle ci soit héritée. En effet cette possibilité permettrait de créer un nouveau constructeur dans la classe fille ou de rendre celle ci clonable.

50.2. Les modèles de structuration

50.3. Les modèles de comportement

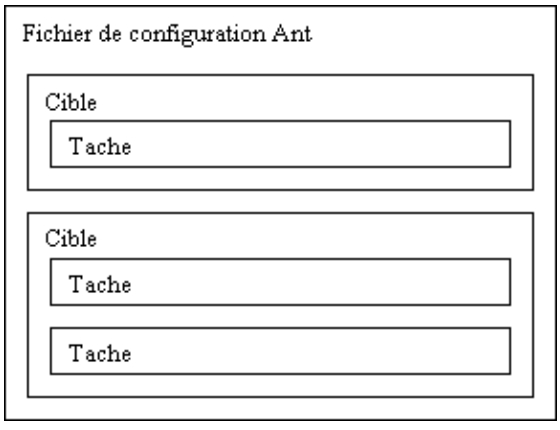
51. Ant

Ant est un projet du groupe Apache−Jakarta. Son but est de fournir un outil écrit en Java pour permettre la construction d'applications (compilation, exécution de tâches post et pré compilation ...). Ces processus de construction d'applications sont très importants car ils permettent d'automatiser des opérations répétitives tout au long du cycle de développement de l'application (développement, tests, recettes, mises en production ...). Le site officiel de l'outil Ant est http://jakarta.apache.org/ant/index.html.

Ant pourrait être comparé au célèbre outil make sous Unix. Il a été développé pour fournir un outil de construction indépendant de toute plate−forme. Ceci est particulièrement utile pour des projets développés sur et pour plusieurs systèmes ou pour migrer des projets d'un système sur un autre. Il est aussi très efficace pour de petits développements.

Ant repose sur un fichier de configuration XML qui décrit les différentes tâches qui devront être exécutées par l'outil. Ant fournit un certain nombre de tâches courantes qui sont codées sous forme d'objets développés en Java. Ces tâches sont donc indépendantes du système sur lequel elles seront exécutées. De plus, il est possible d'ajouter ces propres tâches en écrivant de nouveaux objets Java respectant certaines spécifications.

Le fichier de configuration contient un ensemble de cibles (target). Chaque cible contient une ou plusieurs tâches. Chaque cible peut avoir une dépendance envers une ou plusieurs autres cibles pour pouvoir être exécutée.



Les environnements de développement intégrés proposent souvent un outil de construction propriétaire qui son généralement moins souple et moins puissant que Ant. Ainsi des plug−ins ont été développés pour la majorité d'entre eux (JBuilder, Forte, Visual Age ...) pour leur permettre d'utiliser Ant, devenu un standard de fait.

Ant possède donc plusieurs atouts : multi plate−forme, configurable grâce à un fichier XML, open source et extensible.

Pour obtenir plus de détails sur l'utilisation de Ant, il est possible de consulter la documentation de la version courante à l'url suivante : http://jakarta.apache.org/ant/manual/index.html

Une version 2 de Ant est en cours de développement.

Ce chapitre contient plusieurs sections :

51.1. Installation de Ant

Pour pouvoir utiliser Ant, il faut avoir un JDK 1.1 ou supérieur et installer Ant sur la machine.

51.1.1. Installation sous Windows

Le plus simple est de télécharger la distribution binaire de Ant pour Windows : jakarta−ant−version−bin.zip sur le site de Ant.

Il suffit ensuite de :

- dézipper le fichier (un répertoire jakarta−ant−version est créer, contenant l'outil et sa documentation)
- ajouter le chemin complet au répertoire bin de Ant à la variable système PATH (pour pouvoir facilement appeler Ant n'importe ou dans l'arborescence du système)
- s'assurer que la variable JAVA_HOME pointe sur le répertoire contenant le JDK
- créer une variable d'environnement ANT_HOME qui pointe sur le répertoire jakarta−ant−version créé lors de la décompression du fichier
- il peut être nécessaire d'ajouter les fichiers .jar contenus dans le répertoire lib de Ant à la variable d'environnement CLASSPATH

Exemple de lignes contenues dans le fichier autoexec.bat :

```
...
set JAVA_HOME=c:\jdk1.3 set
ANT_HOME=c:\java\ant
set PATH=%PATH%;%ANT_HOME%\bin
...
```

51.2. Exécuter ant

Ant s'utilise en ligne de commande avec la syntaxe suivante :

ant [options] [cible]

Par défaut, Ant recherche un fichier nommé build.xml dans le répertoire courant. Ant va alors exécuter la cible par défaut définie dans le projet de ce fichier build.xml.

Il est possible de préciser le nom du fichier de configuration en utilisant l'option −buildfile et en la faisant suivre du nom du fichier de configuration.

Exemple :

```
ant −buildfile monbuild.xml
```

Il est possible de préciser une cible précise à exécuter. Dans ce cas, Ant exécute les cibles dont dépend la cible précisée et exécute cette dernière.

Exemple : exécuter la cible clean et toutes les cibles dont elle dépend
`ant clean`

Ant possède plusieurs options dont voici les principales :

Option	Rôle
−quiet	fourni un minimum d'informations lors de l'exécution
− verbose	fourni un maximum d'informations lors de l'exécution
−version	affiche la version de ant
−projecthelp	affiche les cibles définis avec leur description
−buildfile	permet de préciser le nom du fichier de configuration
−Dnom=valeeur	permet de définir une propriété dont le nom et le valeur sont séparés par un caractère =

51.3. Le fichier build.xml

Le fichier build est un fichier XML qui contient la description du processus de construction de l'application.

Comme tout document XML, le fichier débute par un prologue :

<?xml version="1.0">

L'élément principal de l'arborescence du document est le projet représenté par le tag <project> qui est donc le tag racine du document.

A l'intérieur du projet, il faut définir les éléments qui le compose :

- les cibles (targets) : ce sont des étapes du projet de construction
- les propriétés (properties) : ce sont des variables qui contiennent des valeurs utilisables par d'autres éléments (cibles ou tâches)
- les tâches (tasks) : ce sont des traitements unitaires à réaliser dans une cible donnée

Pour permettre l'exécution sur plusieurs plate−formes, les chemins de fichiers utilisés dans le fichier build.xml doivent utiliser la caractère slash '/' comme séparateur même sous Windows qui utilise le caractère anti−slash '\'.

51.3.1. Le projet

Il est défini par le tag racine <project> dans le fichier build.

Ce tag possède plusieurs attributs :

- name : cet attribut précise le nom du projet
- default : cet attribut précise la cible par défaut à exécuter si aucune cible n'est précisée lors de l'exécution
- basedir : cet attribut précise le répertoire qui servira de référence pour l'utilisation de localisation relative des autres répertoires.

Exemple :
`<project name="mon projet" default="compile" basedir=".">`

51.3.2. Les commentaires

Les commentaires sont inclus dans un tag <!−− −−>.

Exemple :

```
<!−− Exemple de commentaires −−>
```

51.3.3. Les propriétés

Le tag <property> permet de définir une propriété qui pourra être utilisée dans le projet : c'est souvent la définition d'un répertoire ou d'une variable qui sera utilisée par certaines tâches. Leur définition en tant que propriété permet de facilement changer leur valeur une seule fois même si la valeur de la propriété est utilisée plusieurs fois dans le projet.

Exemple :

```
<property name= "nom_appli" value= "monAppli"/>
```

Les propriétés sont immuables et peuvent être définies de deux manières :

- avec le tag <property>
- avec l'option −D sur la ligne de commande lors de l'appel de la commande ant

Pour utiliser une propriété sur la ligne de commande, il faut utiliser l'option −D immediatement suivi du nom de la propriété, suivi du caractère =, suivi de la valeur, le tout sans espace.

Le tag <property> possède plusieurs attributs :

- name : cet attribut défini le nom de la propriété
- value : cet attribut défini la valeur de la propriété
- location : cet attribut permet de définir un fichier avec son chemin absolu. Il peut être utilisé à la place de l'attribut value
- file : cet attribut permet de préciser le nom d'un fichier qui contient la définition d'un ensemble de propriétés. Ce fichier sera lu et les propriétés qu'il contient seront définies.

L'utilisation de l'attribut file est particulièrement utile car il permet de séparer la définition des propriétés du fichier build. Le changement d'un paramètre ne nécessite alors pas de modifications dans le fichier xml build.

Exemple :

```
<property file="mesproprietes.properties" />
<property name="repSources" value="src" />
<property name="projet.nom" value="mon_projet" />
<property name="projet.version" value="0.0.10" />
```

L'ordre de définition d'une propriété est très important : Ant gère une priorité sur l'ordre de définition d'une propriété. La règle est la suivante : la première définition d'une propriété est prise en compte, les suivantes sont ignorées.

Ainsi, les propriétés définies via la ligne de commande sont prioritaires par rapport à celles définies dans le fichier build. Il est aussi préférable de mettre le tag <property> contenant un attribut file avant les tag <property> définissant des variables.

Pour utiliser une propriété définie dans le fichier, il faut utiliser la syntaxe suivante :
${nom_propriete}

Exemple :
`${repSources}`

Il existe aussi des propriétés pré−définies par Ant et utilisables dans chaque fichier build :

Propriété	Rôle
basedir	chemin absolu du répertoire de travail (cette valeur est précisée dans l'attribut basedir du tag project)
ant.file	chemin absolu du fichier build en cours de traitement
ant.java.version	version de la JVM qui exécute ant
ant.project.name	nom du projet en cours d'utilisation

51.3.4. Les ensembles de fichiers

Le tag <fileset> permet de définir un ensemble de fichiers. Cet ensemble de fichier sera utilisé dans une autre tâche. La définition d'un tel ensemble est réalisé grace à des attributs du tag <fileset> :

Attribut	Rôle
dir	Définit le répertoire de départ de l'ensemble de fichiers
includes	Liste des fichiers à inclure
excludes	Liste des fichiers à exclure

L'expression **/ permet de désigner tous les sous répertoires du répertoire défini dans l'attribut dir.

Exemple :
`<fileset dir="src" includes="**/*.java">`

51.3.5. Les ensembles de motifs

Le tag <patternset> permet de définir un ensemble de motifs pour sélectionner des fichiers.

La définition d'un tel ensemble est réalisée grace à des attributs du tag <patternset> :

Attribut	Rôle
id	Définit un identifiant pour l'ensemble qui pourra ainsi être réutilisé
includes	Liste des fichiers à inclure
excludes	Liste des fichiers à exclure
refid	Demande la réutilisation d'un ensemble dont l'identifiant est fourni comme valeur

L'expression **/ permet de désigner tous les sous répertoires du répertoire définit dans l'attribut dir. Le caractère ? représente un unique caractère quelconque et le caractère * représente zéro ou n caractères quelconques.

Exemple :

```
<fileset dir="src">
  <patternset id="source_code">
    <includes="**/*.java"/>
  </patternset>
</fileset>
```

51.3.6. Les listes de fichiers

Le tag <filelist> permet de définir une liste de fichiers finis. Chaque fichier est nommément ajouté dans laliste, séparé chacun par une virgule. La définition d'un tel élément est réalisée grace à des attributs du tag <filelist> :

Attribut	Rôle
id	Définit un identifiant pour la liste qui pourra ainsi être réutilisé
dir	Définit le répertoire de départ de la liste de fichiers
files	liste des fichiers séparés par une virgule
refid	Demande la réutilisation d'une liste dont l'identifiant est fourni comme valeur

Exemple :

```
<filelist dir="texte" files="fichier1.txt,fichier2.txt" />
```

51.3.7. Les éléments de chemins

Le tag <pathelement> permet de définir un élément qui sera ajouter à la variable classpath. La définition d'un tel élément est réaliser grace à des attributs du tag <pathelement> :

Attribut	Rôle
location	Défini un chemin d'une ressource qui sera ajoutée
path	

Exemple :

```
<classpath>
  <pathelement location="bin/mabib.jar">
  <pathelement location="lib/">
</classpath>
```

Il est préférable pour assurer une meilleure compatibilité entre plusieurs systèmes d'utiliser des chemins relatifs par rapport au répertoire de base de projet.

51.3.8. Les cibles

Le tag <target> définit une cible. Une cible est un ensemble de tâches à réaliser dans un ordre précis. Cet ordre correspond à celui des tâches décrites dans la cible.

Le tag possède plusieurs attributs :

- name : contient le nom de la cible. Cet attribut est obligatoire
- description : contient une brève description de la cible. Cet attribut est optionnel mais il est recommandé de l'utiliser car la plupart des IDE l'affiche lors de l'utilisation de ant
- if : permet de conditionner l'exécution par l'existence d'une propriété. Cet attribut est optionnel

- unless : permet de conditionner l'exécution par l'inexistance de la définition d'une propriété. Cet attribut est optionnel
- depends : permet de définir la liste des cibles dont dépend la cible. Cet attribut est optionnel

Il est possible de faire dépendre une cible d'une ou plusieurs autres cibles du projet. Lorsqu'une cible doit être exécutée, Ant s'assure que les cibles dont elle dépend ont été complétement exécutées préalablement depuis l'exécution de Ant. Une dépendance est définie grace à l'attribut depends. Plusieurs cibles dépendantes peuvent être listées dans l'attribut depends. Dans ce cas, chaque cible doit être séparée avec une virgule.

51.4. Les tâches (task)

Une tâche est une unité de traitements contenue dans une classe Java qui implémente l'interface org.apache.ant.Task. Dans le fichier de configuration, une tâche est un tag qui peut avoir des paramètres pour configurer le traitement à réaliser. Une tâche est obligatoirement incluse dans une cible.

Ant fournit en standard un certain nombre de tâches pour des traitements courants lors du développement en Java :

Catégorie	Nom de la tâche	Rôle
Tâches internes	echo	Afficher un message
	dependset	Définir des dépendances entre fichiers
	taskdef	Définir une tâche externe
	typedef	Définir un nouveau type de données
Gestion des propriétés	available	Définir une propriété si une ressource existe
	condition	Définir une propriété si une condition est vérifiée
	pathconvert	Définir une propriété avec la conversion d'un chemin de fichier spécifique à un OS
	property	Définir une propriété
	tstamp	Initialiser les propriétés DSTAMP, TSTAMP et TODAY avec la date et heure courante
	uptodate	Définir une propriété en comparant la date de modification de fichiers
tâches Java	java	Exécuter une application dans la JVM
	javac	Compiler des sources Java
	javadoc	Générer la documentation du code source
	rmic	Générer les classes stub et skeleton nécessaires à la technologie rmi
	signjar	Signer un fichier jar
Gestion des archives	ear	Créer une archive contenant une application J2EE
	gunzip	Décompresser une archive
	gzip	Compresser dans une archive
	jar	Créer une archive de type jar
	tar	Créer une archive de type tar
	unjar	Décompresser une archive de type jar
	untar	Décompresser une archive de type tar
	unwar	Décompresser une archive de type war

	unzip	Décompresser une archive de type zip
	war	Créer une archive de type war
	zip	Créer une archive de type zip
tâches diverses	apply	Exécuter une commande externe appliquée à un ensemble de fichiers
	cvs	Gérer les sources dans CVS
	cvspass	
	exec	Exécuter une commande externe
	genkey	Générer une clé dans un trousseau de clé
	get	Obtenir une ressource à partir d'une URL
	mail	Envoyer un courrier électronique
	replace	Remplacer une chaîne de caractères par une autre
	sql	Exécuter une requête SQL
	style	Appliquer une feuille de style XSLT à un fichier XML
Gestion des fichiers	chmod	Modifier les droits d'un fichier
	copy	Copier un fichier
	delete	Supprimer un fichier
	mkdir	Créer un répertoire
	move	Déplacer ou renommer un fichier
	touch	Modifier la date de modification du fichier avec la date courante
Gestion de l'exécution de Ant	ant	Exécuter un autre fichier de build
	antcall	Exécuter une cible
	fail	Stopper l'exécution de Ant
	parallel	Exécuter une tâche en parallèle
	record	Enregistrer les traitements de l'exécution dans un fichier journal
	sequential	Exécuter une tâche en séquentielle
	sleep	Faire une pause dans les traitements

Certaines de ces tâches seront détaillées dans les sections suivantes : pour une référence complète de ces tâches, il est nécessaire de consulter la documentation de Ant.

51.4.1. echo

La tâche <echo> permet d'écrire dans un fichier ou d'afficher un message ou des informations durant l'exécution des traitements.

Les données à utiliser peuvent être fournies dans un attribut dédié ou dans le corps du tag <echo>.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
message	Message à afficher
file	Fichier dans lequel le message sera inséré
append	Booléen qui précise si le message est ajouté à la fin du fichier (true) ou si le fichier doit être écrasé avec le message fourni (false)

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Test avec Ant" default="init" basedir=".">
  <!-- ================================================================= -->
  <!-- Initialisation -->
  <!-- ================================================================= -->
  <target name="init">
    <echo message="Debut des traitements" />
    <echo>
      Fin des traitements du projet ${ant.project.name}
    </echo>
    <echo file="${basedir}/log.txt" append="false" message="Debut des traitements" />
    <echo file="${basedir}/log.txt" append="true" >
Fin des traitements
    </echo>
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
     [echo] Debut des traitements
     [echo]
     [echo]       Fin des traitements du projet Test avec Ant
     [echo]

BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\test\testant>type log.txt
Debut des traitements
Fin des traitements

C:\java\test\testant>
```

51.4.2. mkdir

La tâche <mkdir> permet de créer un répertoire avec éventuellement ses répertoires pères ci ceux ci n'existent pas.

Cette tâche possède un seul attribut:

Attribut	Rôle
dir	Précise le chemin et le nom du répertoire à créer

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Test avec Ant" default="init" basedir=".">
  <!-- ================================================================= -->
  <!-- Initialisation -->
  <!-- ================================================================= -->
  <target name="init">
    <mkdir dir="${basedir}/gen" />
  </target>
```

```
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
    [mkdir] Created dir: C:\java\test\testant\gen

BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\test\testant>
```

51.4.3. delete

La tâche <delete> permet de supprimer des fichiers ou des répertoires.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
file	Permet de préciser le fichier à supprimer
dir	Permet de préciser le répertoire à supprimer
verbose	Booléen qui permet d'afficher la liste des éléments supprimés
quiet	Booléen qui permet de ne pas afficher les messages d'erreurs
includeEmptyDirs	Booléen qui permet de supprimer les répertoires vides

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Test avec Ant" default="init" basedir=".">
  <!-- ================================================================= -->
  <!-- Initialisation -->
  <!-- ================================================================= -->
  <target name="init">
    <delete dir="${basedir}/gen" />
    <delete file="${basedir}/log.txt" />
    <delete>
      <fileset dir="${basedir}/bin" includes="**/*.class" />
    </delete>
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
   [delete] Deleting directory C:\java\test\testant\gen
   [delete] Deleting: C:\java\test\testant\log.txt

BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\test\testant>
```

51.4.4. copy

La tâche <copy> permet de copier un ou plusieurs fichiers dans le cas ou ils n'existent pas dans la cible ou si ils sont plus récents dans la cible.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
file	Désigne le fichier à copier
todir	Permet de préciser le répertoire cible dans lequel les fichiers seront copiés
overwrite	Booléen qui permet d'écraser les fichiers cibles si ils sont plus récents (false par défaut)

L'ensemble des fichiers concernés par la copie doit être précisé avec un tag fils <fileset>.

Exemple :

```
<project name="utilisation de hbm2java" default="init" basedir=".">

  <!-- Definition des proprietes du projet -->
  <property name="projet.sources.dir"    value="src"/>
  <property name="projet.bin.dir"        value="bin"/>

  <!-- Initialisation des traitements -->
  <target name="init" description="Initialisation">
    <!-- Copie des fichiers de mapping et parametrage -->
    <copy todir="${projet.bin.dir}" >
      <fileset dir="${projet.sources.dir}" >
        <include name="**/*.properties"/>
        <include name="**/*.hbm.xml"/>
        <include name="**/*.cfg.xml"/>
      </fileset>
    </copy>
  </target>
</project>
```

Résultat :

```
C:\java\test\testhibernate>ant
Buildfile: build.xml

init:
     [copy] Copying 3 files to C:\java\test\testhibernate\bin

BUILD SUCCESSFUL
Total time: 3 seconds
```

51.4.5. tstamp

La tâche <tstamp> permet de définir trois propriétés :

- DSTAMP : cette propriété est initialisée avec la date du jour au format AAAMMJJ
- TSTAMP : cette propriété est initialisée avec l'heure actuelle sous la forme HHMM
- TODAY : cette propriété est initialisée avec la date du jour au format long

Cette tâche ne possède pas d'attributs.

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Test avec Ant" default="init" basedir=".">
  <!-- ================================================================= -->
  <!-- Initialisation -->
```

```
  <!-- ================================================================== -->
  <target name="init">
    <tstamp/>
    <echo message="Nous sommes le ${TODAY}" />
    <echo message="DSTAMP = ${DSTAMP}" />
    <echo message="TSTAMP = ${TSTAMP}" />
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
     [echo] Nous sommes le August 25 2004
     [echo] DSTAMP = 20040825
     [echo] TSTAMP = 1413

BUILD SUCCESSFUL
Total time: 2 seconds
```

51.4.6. java

La tâche <java> permet de lancer une machine virtuelle pour exécuter une application compilée.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
classname	nom pleinement qualifié de la classe à exécuter
jar	nom du fichier de l'application à exécuter
classpath	classpath pour l'exécution. Il est aussi possible d'utiliser un tag fils <classpath> pour le spécifier
classpathref	utilisation d'un classpath précédemment défini dans le fichier de build
fork	lancer l'exécution dans une JVM dédiée au lieu de celle ou l'exécute Ant
output	enregistrer les sorties de la console dans un fichier

La tag fils <arg> permet de fournir des paramètres à l'exécution.

Le tag fils <classpath> permet de définir le classpath à utiliser lors de l'exécution

Exemple :

```
<project name="testhibernate1" default="TestHibernate1" basedir=".">

  <!-- Definition des proprietes du projet -->
  <property name="projet.sources.dir"    value="src"/>
  <property name="projet.bin.dir"        value="bin"/>
  <property name="projet.lib.dir"        value="lib"/>

  <!-- Definition du classpath du projet -->
  <path id="projet.classpath">
    <fileset dir="${projet.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${projet.bin.dir}" />
  </path>

  <!-- Execution de TestHibernate1 -->
  <target name="TestHibernate1" description="Execution de TestHibernate1" >
    <java classname="TestHibernate1" fork="true">
```

```
      <classpath refid="projet.classpath"/>
    </java>
  </target>
</project>
```

51.4.7. javac

La tâche <javac> permet la compilation de fichiers source contenus dans une arborescence de répertoires.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
srcdir	précise le répertoire racine de l'arborescence du répertoire contenant les sources
destdir	précise le répertoire où les résultats des compilations seront stockés
classpath	classpath pour l'exécution. Il est aussi possible d'utiliser un tag fils <classpath> pour le spécifier
classpathref	utilisation d'un classpath précédemment défini dans le fichier de build
nowarn	précise si les avertissements du compilateur doivent être affichés. La valeur par défaut est off
debug	précise si le compilateur doit inclure les informations de débogage dans les fichiers compilés. La valeur par défaut est off
optimize	précise si le compilateur doit optimiser le code compilé qui sera généré. La valeur par défaut est off
deprecation	précise si les avertissements du compilateur concernant l'usage d'éléments deprecated doivent être affichés. La valeur par défaut est off
target	précise la version de la plate−forme Java cible (1.1, 1.2, 1.3, 1.4, ...)
fork	lance la compilation dans une JVM dédiée au lieu de celle ou s'exécute Ant. La valeur par défaut est false
failonerror	précise si les erreurs de compilations interrompent l'exécution du fichier de build. La valeur par défaut est true
source	version des sources java : particulièrement utile pour Java 1.4 et 1.5 qui apportent des modifications à la grammaire du langage Java

Exemple :

```
<project name="compiltation des classes" default="compile" basedir=".">
  <!−− Definition des proprietes du projet −−>
  <property name="projet.sources.dir"    value="src"/>
  <property name="projet.bin.dir"        value="bin"/>
  <property name="projet.lib.dir"        value="lib"/>

  <!−− Definition du classpath du projet −−>
  <path id="projet.classpath">
    <fileset dir="${projet.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${projet.bin.dir}" />
  </path>

  <!−− Compilation des classes du projet −−>
  <target name="compile" description="Compilation des classes">
    <javac srcdir="${projet.sources.dir}"
              destdir="${projet.bin.dir}"
              debug="on"
```

```
              optimize="off"
              deprecation="on">
      <classpath refid="projet.classpath"/>
  </javac>
  </target>
</project>
```

Résultat :

```
C:\java\test\testhibernate>ant
Buildfile: build.xml

compile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin
    [javac] C:\java\test\testhibernate\src\TestHibernate1.java:9: cannot resolve
 symbol
    [javac] symbol : class configuration
    [javac] location: class TestHibernate1
    [javac] Configuration config = new configuration();
    [javac] ^
    [javac] 1 error

BUILD FAILED
file:C:/java/test/testhibernate/build.xml:22: Compile failed; see the compiler e
rror output for details.

Total time: 9 seconds
```

51.4.8. javadoc

La tâche <javadoc> permet de demander la génération de la documentation au format javadoc des classes incluses dans une arborescence de répertoires.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
sourcepath	précise le répertoire de base qui contient les sources dont la documentation est à générer
destdir	précise le répertoire qui va contenir les fichiers de documentation générés

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<project name="Test avec Ant" default="javadoc" basedir=".">
  <!-- ================================================-->
  <!-- Génération de la documentation Javadoc          -->
  <!-- ================================================-->
  <target name="javadoc">
    <javadoc sourcepath="src"
                destdir="doc" >
      <fileset dir="src" defaultexcludes="yes">
        <include name="**" />
      </fileset>
    </javadoc>
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

javadoc:
```

```
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source file C:\java\test\testant\src\MaClasse.java...
  [javadoc] Constructing Javadoc information...
  [javadoc] Standard Doclet version 1.4.2_02
  [javadoc] Building tree for all the packages and classes...
  [javadoc] Building index for all the packages and classes...
  [javadoc] Building index for all classes...

BUILD SUCCESSFUL
Total time: 9 seconds
```

51.4.9. jar

La tâche <jar> permet la création d'une archive de type jar.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
jarfile	nom du fichier .jar à créer
basedir	précise de répertoire qui contient les éléments à ajouter dans l'archive
compress	précise si le contenu de l'archive doit être compressé ou non. La valeur par défaut est true
manifest	précise le fichier manifest qui sera utilisé dans l'archive

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <project name="Test avec Ant" default="packaging" basedir=".">

  <!-- ================================================================== -->
  <!-- Génération de l'archive jar                                        -->
  <!-- ================================================================== -->
  <target name="packaging">
    <jar jarfile="test.jar" basedir="src" />
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

packaging:
      [jar] Building jar: C:\java\test\testant\test.jar

BUILD SUCCESSFUL
Total time: 2 seconds
```

La suite de ce chapitre sera développée dans une version future de ce document

52. Maven

Maven permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java.

Le site officiel est http://maven.apache.org

Il permet notamment :

- d'automatiser certaines tâches : compilation, tests unitaires et déploiement des applications qui composent le projet
- de gérer des dépendances vis à vis des bibliothèques nécessaires au projet
- de générer des documentations concernant le projet

Au premier abord, il est facile de croire que Maven fait double emploi avec Ant. Ant et Maven sont tous les deux développés par le groupe Jakarta, ce qui prouve bien que leur utilité n'est pas aussi identique.

Ant dont le but est d'automatiser certaines tâches répétitives est plus ancien que Maven. Maven propose non seulement ces fonctionnalités mais en propose de nombreuses autres.

Pour gérer les dépendances du projet vis à vis de bibliothèques, Maven utilise un ou plusieurs repositorys qui peuvent être locaux (.maven/repository) ou distants (http://www.ibiblio.org/maven par défaut)

Maven est extensible grâce à un mécanisme de plug in qui permet d'ajouter des fonctionnalités.

52.1. Installation

Il faut télécharger le fichier maven−1.0−rc2.exe sur le site de Maven et l'exécuter.

Un assistant permet de fournir les informations concernant l'installation :

- sur la page « Licence Agreement » : lire la licence et si vous l'acceptez cliquer sur le bouton « I Agree ».
- sur la page « Installations Options » : sélectionner les éléments à installer et cliquer sur le bouton « Next ».
- sur la page « Installation Folder » : sélectionner le répertoire dans lequel Maven va être installé et cliquer sur le bouton « Install ».
- une fois les fichiers copiés, il suffit de cliquer sur le bouton « Close ».

Sous Windows, un élément de menu nommé « Apache Software Foundation / Maven 1.0−rc2 » est ajouté dans le menu « Démarrer / Programmes ».

Pour utiliser Maven, la variable d'environnement système nommée MAVEN_HOME doit être définie avec comme valeur le chemin absolu du répertoire dans lequel Maven est installé. Par défaut, cette variable est configurée automatiquement lors de l'installation sous Windows.

Il est aussi particulièrement pratique d'ajouter le répertoire %MAVEN_HOME%/bin à la variable d'environnement PATH. Maven étant un outil en ligne de commande, cela évite d'avoir à saisir son chemin complet lors de son exécution.

Enfin, il faut créer un repository local en utilisant la commande ci dessous dans une boîte de commandes DOS :

Exemple :

```
C:\>install_repo.bat %HOMEDRIVE%%HOMEPATH%
```

Pour s'assurer de l'installation correcte de Maven, il suffit de saisir la commande :

Exemple :

```
C:\>maven −v
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / −_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0−rc2
C:\>
```

Lors de la première exécution de Maven, ce dernier va constituer le repository local (une connexion internet est nécessaire).

Exemple :

```
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / −_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0−rc2
Le rÚpertoire C:\Documents and Settings\Administrateur\.maven\repository n'exist
e pas. Tentative de crÚation.
Tentative de tÚlÚchargement de commons−lang−1.0.1.jar.
.............................
.
Tentative de tÚlÚchargement de commons−net−1.1.0.jar.
.....................................................................................
........
.
Tentative de tÚlÚchargement de dom4j−1.4−dev−8.jar.
.....................................................................................
.....................................................................................
.....................................................................................
.........................................................
.
Tentative de tÚlÚchargement de xml−apis−1.0.b2.jar.
.....................................................
```

52.2. Les plug−ins

Toutes les fonctionnalités de Maven sont proposées sous la forme de plug−ins.

Le fichier maven.xml permet de configurer les plug−ins installés.

52.3. Le fichier project.xml

Maven est orienté projet, donc le projet est l'entité principale gérée par Maven. Il est nécessaire de fournir à Maven une description du projet (Project descriptor) sous la forme d'un document XML nommé project.xml et situé à la racine du répertoire contenant le projet.

Exemple : un fichier minimaliste

```
<project>

   <id>P001</id>
```

```
    <name>TestMaven</name>
    <currentVersion>1.0</currentVersion>
    <shortDescription>Test avec Maven</shortDescription>

    <developers>
      <developer>
        <name>Jean Michel D.</name>
        <id>jmd</id>
        <email>jmd@test.fr</email>
      </developer>
    </developers>

    <organization>
      <name>Jean-Michel</name>
    </organization>

  </project>
```

Il est possible d'inclure la valeur d'un tag défini dans le document dans un autre tag.

Exemple :

```
 ...
  <shortDescription>${pom.name} est un test avec Maven</shortDescription>
  ...
```

Il est possible d'hériter d'un fichier project.xml existant dans lequel des caractéristiques communes à plusieurs projets sont définies. La déclaration dans le fichier du fichier père se fait avec le tag <extend>. Dans le fichier fils, il suffit de redéfinir ou de définir les tags nécessaires.

52.4. Exécution de Maven

Maven s'utilise en ligne de commande sous la forme suivante :

Maven plugin:goal

Il faut exécuter Maven dans le répertoire qui contient le fichier project.xml.

Si les paramètres fournis ne sont pas corrects, une exception est levée :

Exemple :

```
C:\java\test\testmaven>maven compile
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0-rc2
com.werken.werkz.NoSuchGoalException: No goal [compile]
        at com.werken.werkz.WerkzProject.attainGoal(WerkzProject.java:190)
        at org.apache.maven.plugin.PluginManager.attainGoals(PluginManager.java:
531)
        at org.apache.maven.MavenSession.attainGoals(MavenSession.java:265)
        at org.apache.maven.cli.App.doMain(App.java:466)
        at org.apache.maven.cli.App.main(App.java:1117)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:324)
        at com.werken.forehead.Forehead.run(Forehead.java:551)
        at com.werken.forehead.Forehead.main(Forehead.java:581)
Total time: 4 seconds
Finished at: Tue May 18 14:17:18 CEST 2004
```

Pour obtenir une liste complète des plug−ins à disposition de Maven, il suffit d'utiliser la commande maven –g

Voici quelques uns des nombreux plug−in avec leurs goals principaux :

Plug in	Goal	Description
ear	ear	construire une archive de type ear
	deploy	déployer un fichier ear dans un serveur d'application
ejb	ejb	
	deploy	
jalopy	format	
java	compile	compiler des sources
	jar	créer une archive de type .jar
javadoc		
jnlp		
pdf		générer la documentation du projet au format PDF
site	generate	générer le site web du projet
	deploy	copier le site web sur un serveur web
test	match	exécuter des tests unitaires
war	init	
	war	
	deploy	

La commande maven clean permet d'effacer tous les fichiers générés par Maven.

Exemple :

```
C:\java\test\testmaven>maven clean
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / −_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0−rc2
build:start:
clean:clean:
    [delete] Deleting directory C:\java\test\testmaven\target
    [delete] Deleting: C:\java\test\testmaven\velocity.log
BUILD SUCCESSFUL
Total time: 2 minutes 7 seconds
Finished at: Tue May 25 14:19:03 CEST 2004
C:\java\test\testmaven>
```

52.5. Génération du site du projet

Maven propose une fonctionnalité qui permet de générer automatique un site web pour le projet regroupant un certain nombre d'informations utiles le concernant.

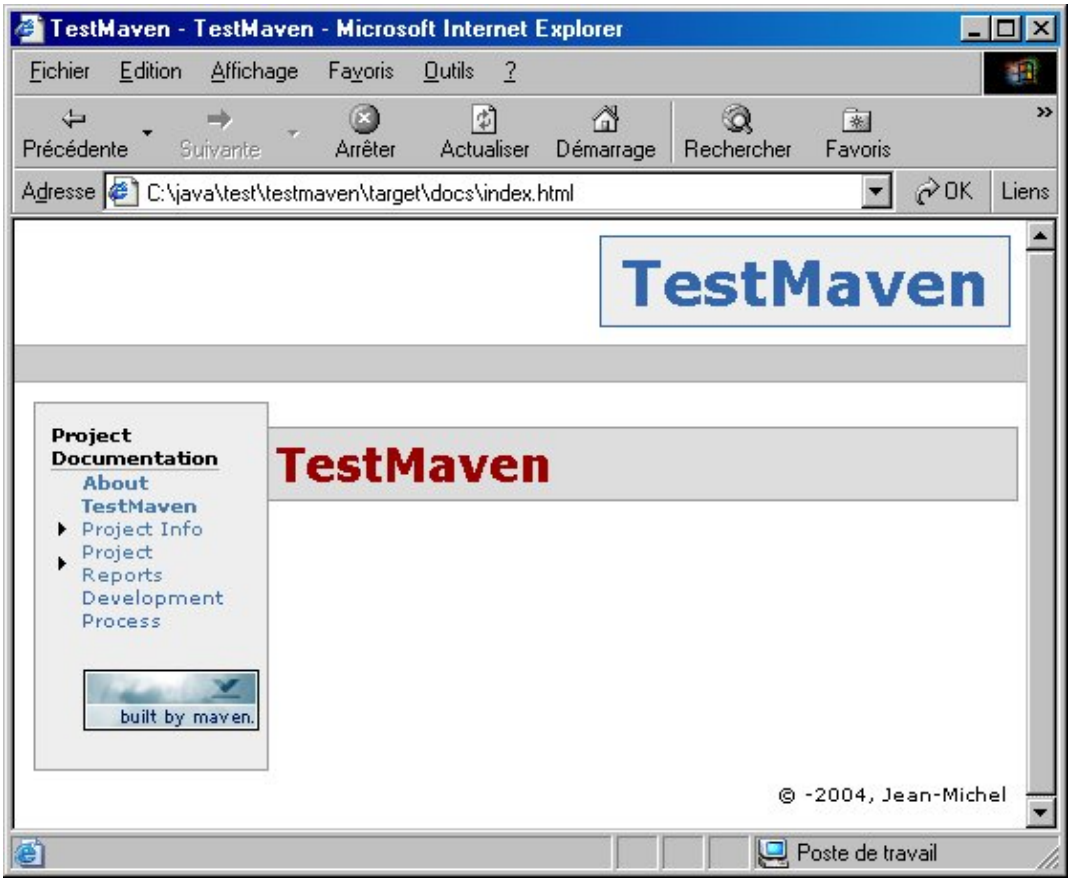
Pour demander la génération du site, il suffit de saisir la commande

maven site:generate

Lors de l'exécution de cette commande, un répertoire target/docs est créé contenant les différents éléments du site.

Exemple :

```
C:\java\test\testmaven\target\docs>dir
 Le volume dans le lecteur C s'appelle MACHINE
 Le numéro de série du volume est 3T78-19E4
 Répertoire de C:\java\test\testmaven\target\docs
25/05/2004  14:21       <DIR>          .
25/05/2004  14:21       <DIR>          ..
25/05/2004  14:21       <DIR>          apidocs
25/05/2004  14:21                5 961 checkstyle-report.html
25/05/2004  14:21                1 637 cvs-usage.html
25/05/2004  14:21                1 954 dependencies.html
25/05/2004  14:21       <DIR>          images
25/05/2004  14:21                1 625 index.html
25/05/2004  14:21                1 646 issue-tracking.html
25/05/2004  14:21                3 119 javadoc.html
25/05/2004  14:21                9 128 jdepend-report.html
25/05/2004  14:21                2 494 license.html
25/05/2004  14:21                5 259 linkcheck.html
25/05/2004  14:21                1 931 mail-lists.html
25/05/2004  14:21                4 092 maven-reports.html
25/05/2004  14:21                3 015 project-info.html
25/05/2004  14:21       <DIR>          style
25/05/2004  14:21                2 785 task-list.html
25/05/2004  14:21                3 932 team-list.html
25/05/2004  14:21       <DIR>          xref
              14 fichier(s)           48 578 octets
               6 Rép(s)     207 151 616 octets libres
```



Par défaut, le site généré contient un certain nombre de pages accessibles via le menu de gauche.

La partie « Project Info » regroupe trois pages : la mailing liste, la liste des développeurs et les dépendances du projet.

La partie « Project report » permet d'avoir accès à des comptes rendus d'exécution de certaines tâches : javadoc, tests unitaires, ... Certaines de ces pages ne sont générées qu'en fonction des différents éléments générés par Maven.

Le contenu du site pourra donc être réactualisé facilement en fonction des différents traitements réalisés par Maven sur le projet.

52.6. Compilation du projet

Dans le fichier project.xml, il faut rajouter un tag <build> qui va contenir les informations pour la compilation des éléments du projet.

Les sources doivent être contenues dans un répertoire dédié, par exemple src

Exemple :

```
 ...
    <build>

        <sourceDirectory>

          ${basedir}/src

        </sourceDirectory>

      </build>
  ...
```

Pour demander la compilation à Maven, il faut utiliser la commande

Exemple :

```
Maven java :compile
C:\java\test\testmaven>maven java:compile
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0-rc2
Tentative de tÚlÚchargement de commons-jelly-tags-antlr-20030211.143720.jar.
.....
.
build:start:
java:prepare-filesystem:
    [mkdir] Created dir: C:\java\test\testmaven\target\classes
java:compile:
    [echo] Compiling to C:\java\test\testmaven/target/classes
    [javac] Compiling 1 source file to C:\java\test\testmaven\target\classes
BUILD SUCCESSFUL
Total time: 12 seconds
Finished at: Tue May 18 14:19:12 CEST 2004
```

Le répertoire « target/classes » est créé à la racine du répertoire du projet. Les fichiers .class issus de la compilation sont stockés dans ce répertoire.

La commande maven jar permet de demander la génération du packaging de l'application.

Exemple :

```
build:start:
java:prepare-filesystem:
java:compile:
    [echo] Compiling to C:\java\test\testmaven/target/classes
java:jar-resources:
test:prepare-filesystem:
    [mkdir] Created dir: C:\java\test\testmaven\target\test-classes
    [mkdir] Created dir: C:\java\test\testmaven\target\test-reports
test:test-resources:
```

```
test:compile:
    [echo] No test source files to compile.
test:test:
    [echo] No tests to run.
jar:jar:
    [jar] Building jar: C:\java\test\testmaven\target\P001-1.0.jar
BUILD SUCCESSFUL
Total time: 2 minutes 42 seconds
Finished at: Tue May 18 14:25:39 CEST 2004
```

Par défaut, l'appel à cette commande effectue une compilation des sources, un passage des tests unitaires si il y en a et un appel à l'outil jar pour réaliser le packaging.

Le nom du fichier jar créé est composé de l'id du projet et du numéro de version. Il est stocké dans le répertoire racine du projet.

53. Les frameworks de tests

L'utilisation de frameworks dédiés à l'automatisation des test unitiaires permet d'assurer une meilleure qualité et fiabilité du code. Cette automatisation facilite aussi le passage de tests de non regression notamment lors des nombreuses mises à jour du code. De plus, l'utilisation de ces frameworks ne nécessite aucune modification dans le code à tester ce qui sépare clairement les traitements représentés dans le code de leur test.

Ce chapitre contient plusieurs sections :

- JUnit
- Cactus

53.1. JUnit

JUnit est un framework open source pour réaliser des tests unitaires sur du code Java. Le principal intérêt est de s'assurer que le code répond toujours au besoin même après d'éventuelles modifications. Plus généralement, ce type de tests est appellé tests unitaires de non régression.

Il a été initialement développé par Erich Gamma and Kent Beck.

Le but est d'automatiser les tests. Ceux ci sont exprimés dans des classes sous la forme de cas de tests avec leurs résultats attendus. JUnit exécute ces tests et les comparent avec ces résultats.

Cela permet de séparer le code de la classe, du code qui permet de la tester. Souvent pour tester une classe, il est facile de créer une méthode main() qui va contenir les traitements de tests. L'inconvénient est que ce code "superflu" aux traitements proprement dit est qu'il soit inclus dans la classe. De plus, son exécution doit se faire à la main.

Avec Junit, l'unité de test est une classe dédiée qui regroupe des cas de tests. Ces cas de tests exécutent les tâches suivantes :

- création d'une instance de la classe et de tout autre objet nécessaire aux tests
- appel de la méthode à tester avec les paramètres du cas de test
- comparaison du résultat obtenu avec le résultat attendu : en cas d'échec, une exception est levée

JUnit est particulièrement adapté pour être utilisé avec la méthode eXtreme Programming puisque de cette méthode préconise entre autre l'automatisation des tâches de tests unitaires qui ont été définis avant l'écriture du code.

La dernière version de JUnit peut être téléchargée sur le site www.junit.org. Pour l'installer, il suffit de dézipper l'archive. Le plus simple pour pouvoir utiliser JUnit est de copier le fichier junit.jar dans le répertoire JAVA_HOME\jre\lib\ext .

La version utilisée dans ce chapitre est la 3.8.1.

53.1.1. Un exemple très simple

L'exemple utilisé dans cette section est la classe suivante :

Exemple :

```
public class MaClasse{

  public static int calculer(int a, int b) {
    int res = a + b;

    if (a == 0){
      res = b * 2;
    }

    if (b == 0) {
      res = a * a;
    }
    return res;
  }

}
```

Il faut compiler cette classe : javac MaClasse.java

Il faut ensuite écrire une classe qui va contenir les différents tests à réaliser par JUnit. L'exemple est volontairement simpliste en ne définissant qu'un seul cas de test.

Exemple :

```
import junit.framework.*;

public class MaClasseTest extends TestCase{

  public void testCalculer() throws Exception {

    assertEquals(2,MaClasse.calculer(1,1));
  }

}
```

Il faut compiler cette classe : javac MaClasseTest.java (le fichier junit.jar doit être dans le classpath).

Enfin, il suffit d'appeler JUnit pour qu'il exécute la séquence de tests.

Exemple :

```
java −cp junit.jar;. junit.textui.TestRunner MaClasseTest
C:\java\testjunit>java −cp junit.jar;. junit.textui.TestRunner
  MaClasseTest
  .
  Time: 0,01
OK (1 test)
```

Attention : le respect de la casse dans le nommage des méthodes de tests est très important. Les méthodes de tests doivent obligatoirement commencer par test en minuscule.

Exemple :

```
import junit.framework.*;

public class MaClasseTest extends TestCase{

  public void TestCalculer() throws Exception {
```

```
    assertEquals(2,MaClasse.calculer(1,1));
  }

}
```

L'utilisation de cette classe avec JUnit produit le résultat suivant :

Exemple :

```
C:\java\testjunit>java -cp junit.jar;. junit.textui.TestRunner
MaClasseTest
.F
Time: 0,01
There was 1 failure:
1) warning(junit.framework.TestSuite$1)junit.framework.AssertionFailedError: No
tests found in MaClasseTest

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

53.1.2. Exécution des tests avec JUnit

JUnit propose trois applications différentes nommées TestRunner pour exécuter les tests :

- une application console : junit.textui.TestRunner.
- une application graphique avec une interface Swing : junit.swingui.TestRunner
- une application graphique avec une interface AWT : junit.awtui.TestRunner

Quelque soit l'application utilisée, le fichier junit.jar doit obligatoirement être inclus dans le CLASSPATH.

L'échec d'un cas de test peut avoir deux origines :

- le cas de test est mal défini
- le code exécuté contient un ou plusieurs bugs.

Suite à l'exécution d'un cas de test, celui ci peut avoir un des trois états suivants :

- échoué : une exception de type AssertionFailedError est levée
- en erreur : une exception non émise par le framework et non capturée a été levée dans les traitements
- passé avec succès

L'échec d'un seul cas de test entraine l'échec du test complet.

53.1.2.1. Exécution des tests dans la console

L'utilisation de l'application console nécessite quelques paramètres lors de son utilisation :

Exemple :

```
C:\>java -cp junit.jar;. junit.textui.TestRunner MaClasseTest
```

Le seul paramètre obligatoire est le nom de la classe de tests. Celle ci doit obligatoirement être sous la forme pleinement qualifiée avec la notation Java si elle appartient à un package.

Exemple :

```
C:\>java -cp junit.jar;. junit.textui.TestRunner com.moi.test.junit.MaClasseTest
```

Il est possible de faire appel au TestRunner dans une application en utilisant sa méthode run() avec en paramètre un objet de type Class qui encapsule la classe de tests à exécuter.

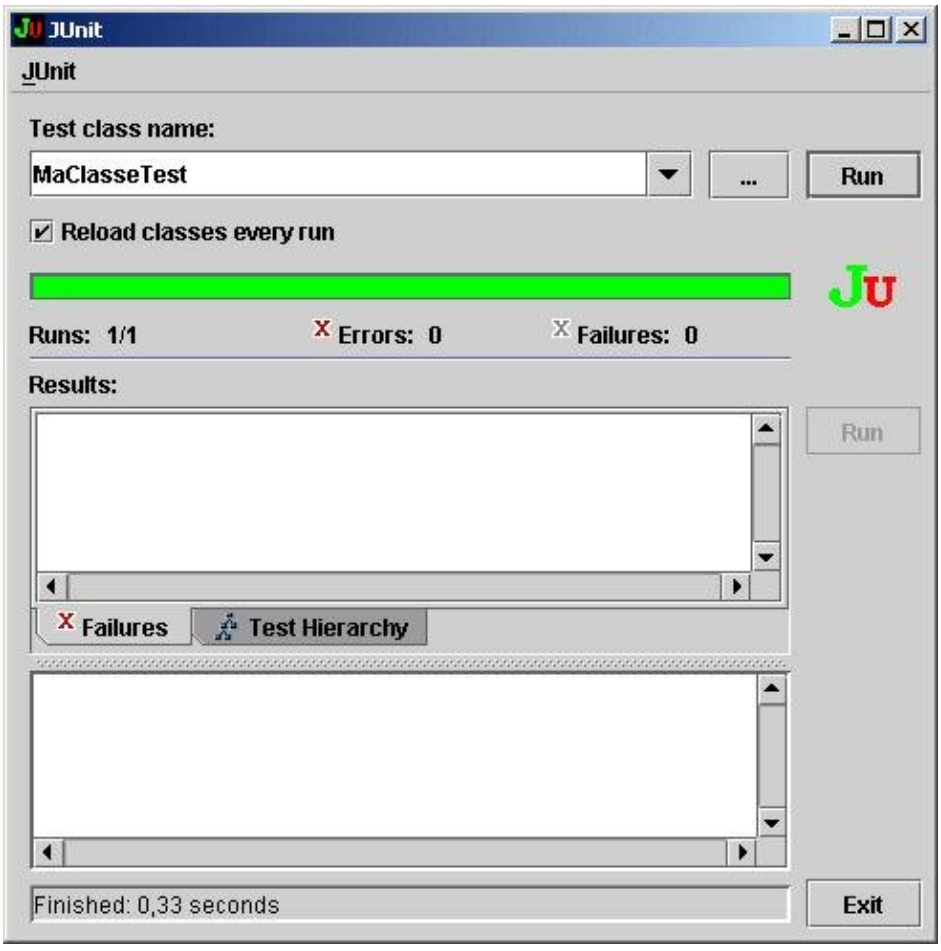
Exemple :

```
public class TestJUnit1 {
    public static void main(String[] args) {
        junit.textui.TestRunner.run(MaClasseTest.class);
    }
}
```

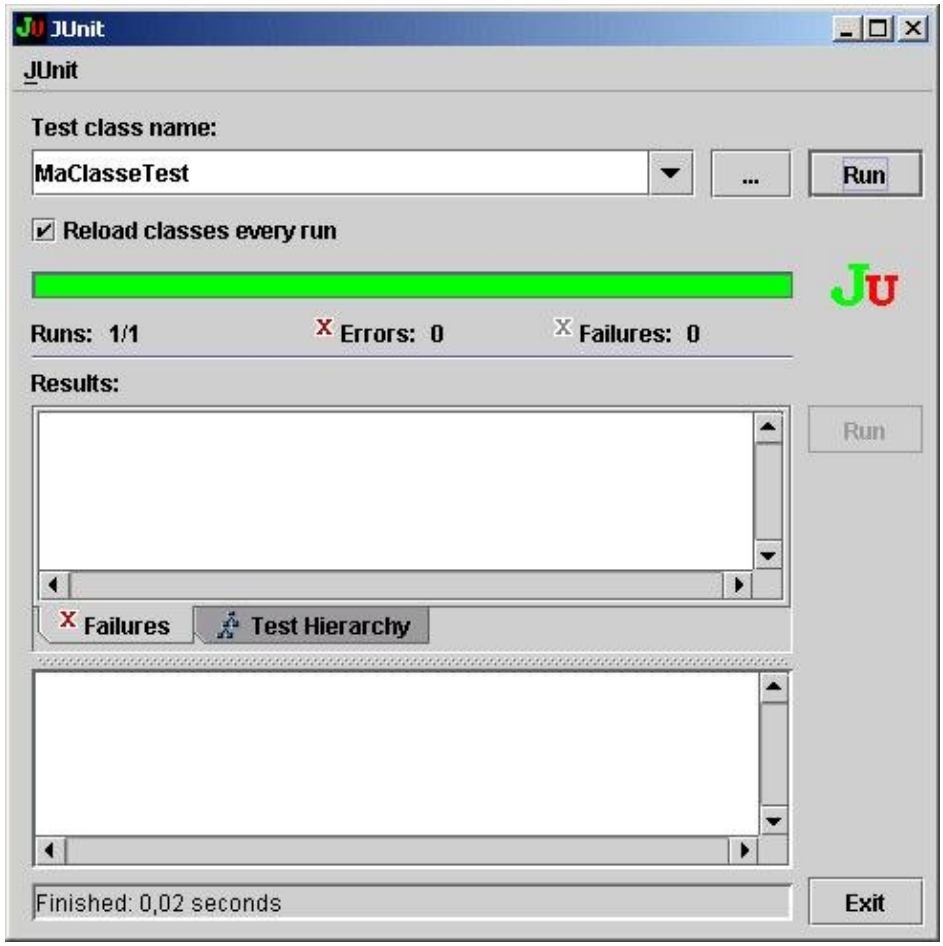
53.1.2.2. Exécution des tests dans une application graphique

Pour utiliser des classes de tests avec ces applications graphiques, il faut obligatoirement que les classes de tests et toutes celles dont elles dépendent soient incluses dans le CLASSPATH. Elles doivent obligatoirement être sous la forme de fichier .class non incluses dans un fichier jar.

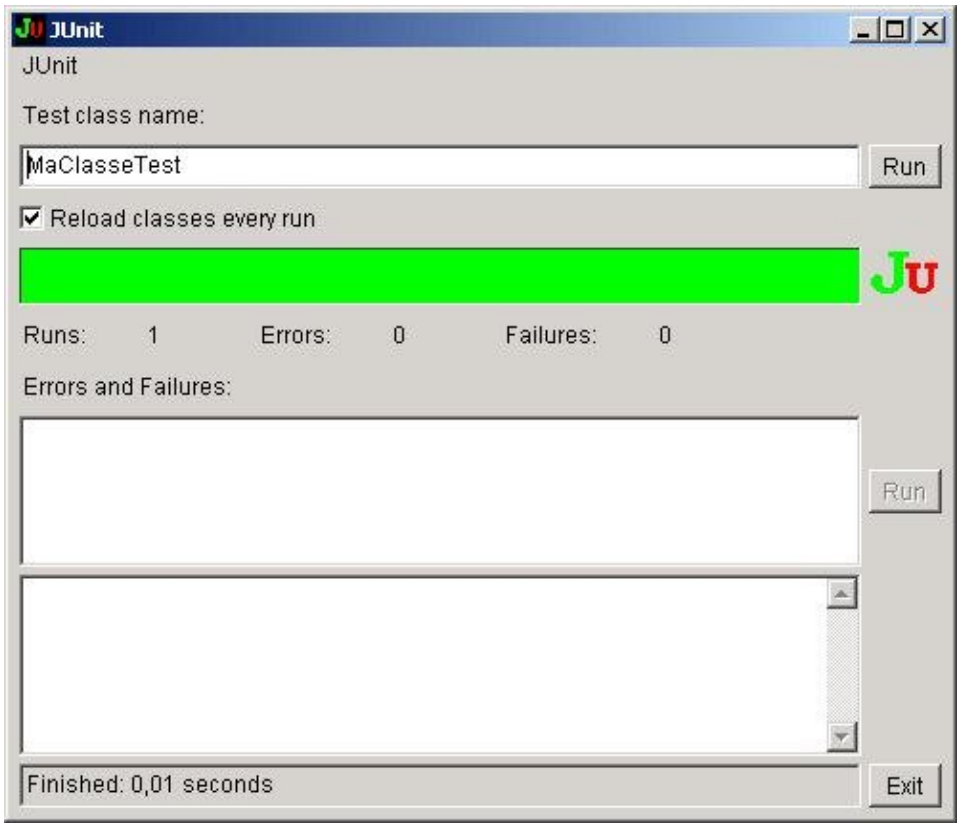
C:\java\testjunit>java -cp junit.jar;. junit.swingui.TestRunner MaClasseTest



Il suffit de cliquer sur le bouton "Run" pour lancer l'exécution des tests.



C:\java\testjunit>java −cp junit.jar;. junit.awtui.TestRunner MaClasseTest



La case à cocher "Reload classes every run" indique à JUnit de recharger les classes à chaque exécution. Ceci est très pratique car cela permet de modifier les classes et de laisser l'application de test ouverte.

53.1.3. Ecriture des cas de tests JUnit

53.1.3.1. Définition de la classe de tests

Pour écrire les cas de tests, il faut écrire une classe qui étende la classe junit.framework.TestCase. Le nom de cette classe est le nom de la classe à tester suivi par "Test".

Exemple :

```
import junit.framework.*;

public class MaClasseTest extends TestCase{

  public void testCalculer() throws Exception {
    fail("Cas de test a ecrire");
  }
}
```

Dans cette classe, il faut écrire une méthode dont le nom commence par "test" en minuscule suivi du nom de chaque méthode de la classe à tester. Chacune de ces méthodes doit avoir les caractèrisques suivantes :

- elle doit être déclarée public
- elle ne doit renvoyer aucune valeur
- elle ne doit pas posséder de paramètres.

Par introspection, JUnit va automatiquement rechercher toutes les méthodes qui respectent cette convention.

Le respect de ces règles est donc important pour une bonne exécution des tests par JUnit.

Exemple : la méthode commence par T et non t

```
import junit.framework.*;

public class MaClasseTest extends TestCase{

  public void TestCalculer() throws Exception {

    // assertEquals(2,MaClasse.calculer(1,1));
    fail("Cas de test a ecrire");
  }

}
```

Résultat :

```
C:\>java -cp junit.jar;. junit.textui.TestRunner
MaClasseTest
.F
Time: 0,01
There was 1 failure:
1) warning(junit.framework.TestSuite$1)junit.framework.AssertionFailedError: No
tests found in MaClasseTest

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

53.1.3.2. Définition des cas de tests

La classe suivante sera utilisée dans les exemples de cette section :

Exemple :

```
public class MaClasse2{
  private int a;
  private int b;

  public MaClasse2(int a, int b) {
    this.a = a;
    this.b = b;
  }

  public int getA() {
    return a;
  }

  public int getB() {
    return b;
  }

  public void setA(int unA) {
    this.a = unA;
  }

  public void setB(int unB) {
    this.b = unB;
  }

  public int calculer() {
    int res = a + b;

    if (a == 0){
      res = b * 2;
    }

    if (b == 0) {
      res = a * a;
    }
    return res;
  }

  public int sommer() throws IllegalStateException {
    if ((a == 0) && (b==0)) {
      throw new IllegalStateException("Les deux valeurs sont nulles");
    }
    return a+b;
  }

}
```

Les cas de tests utilisent des affirmations (assertion en anglais) sous la forme de méthodes nommées assertXXX() proposées par le framework. Il existe de nombreuses méthodes de ce type qui sont héritées de la classe junit.framework.Assert :

Méthode	Rôle
assertEquals()	Vérifier l'égalité de deux valeurs de type primitif. Il existe de nombreuses surcharges de cette méthode pour chaque type primitif, pour un objet de type Object et pour un objet de type String
assertFalse()	Vérifier que la valeur fournie en paramètre est fausse
assertNotNull()	Vérifier que l'objet fourni en paramètre ne soit pas null
assertSame()	Vérifier que les deux objets fournis en paramètre font référence à la même entité
assertTrue()	Vérifier que la valeur fournie en paramètre est vraie

Chacune de ces méthodes possède une version surchargée qui accepte un paramètre supplémentaire sous la forme d'une chaîne de caractères indiquant un message qui sera affiché en cas d'échec du cas de test.

Exemple : un cas de test simple

```
import junit.framework.*;

public class MaClasse2Test extends TestCase{

  public void testCalculer() throws Exception {
    MaClasse2 mc = new MaClasse2(1,1);
    assertEquals(2,mc.calculer());
  }

}
```

L'ordre des paramètres contenant la valeur attendue et la valeur obtenue est important pour obtenir un message d'erreur fiable en cas d'échec du cas de test.

La méthode fail() permet de forcer le cas de test à échouer. Une version surchargée permet de préciser un message qui sera affiché.

Il est aussi souvent utile lors de la définition des cas de tests de devoir tester si une exception est levée lors de l'exécution des traitements.

Exemple :

```
import junit.framework.*;

public class MaClasse2Test extends TestCase{

  public void testSommer() throws Exception {
    MaClasse2 mc = new MaClasse2(0,0);
    mc.sommer();
  }
}
```

Résultat :

```
C:\>java -cp junit.jar;. junit.textui.TestRunner MaClasse2Test
.E
Time: 0,01
There was 1 error:
1) testSommer(MaClasse2Test)java.lang.IllegalStateException: Les deux valeurs so
nt nulles
        at MaClasse2.sommer(MaClasse2.java:42)
        at MaClasse2Test.testSommer(MaClasse2Test.java:31)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)

FAILURES!!!
Tests run: 2,  Failures: 0,  Errors: 1
```

Avec JUnit, pour réaliser de tels cas de tests, il suffit d'appeler la méthode avec les conditions qui doivent lever une exception, d'encapsuler cet appel dans un bloc try/catch et d'appler la méthode fail() si l'exception désirée n'est pas levée.

Exemple :

```
import junit.framework.*;

public class MaClasse2Test extends TestCase{

  public void testSommer() throws Exception {
```

```
    MaClasse2 mc = new MaClasse2(1,1);

    // cas de test 1
    assertEquals(2,mc.sommer());

    // cas de test 2
    try {
      mc.setA(0);
      mc.setB(0);
      mc.sommer();
      fail("Une exception de type IllegalStateException aurait du etre levee");
    } catch (IllegalStateException ise) {
    }

  }
```

53.1.3.3. La création et la destruction d'objets

Il est fréquent que les cas de tests utilisent une instance d'un même objet ou nécessitent l'usage de ressources particulières tel qu'une instance d'une classe pour l'accès à une base de données.

Pour réaliser ces opérations de créations et de destructions d'objets, la classe TestCase contient les méthodes setUp() et tearDown() qui sont respectivement appelées avant et après l'appel de chaque méthode contenant un cas de test.

Il suffit simplement de redéfinir en fonction de ces besoins ces deux méthodes.

53.1.4. Les suites de tests

Les suites de tests permettent de regrouper plusieurs tests dans une même classe. Ceci permet l'automatisation de l'ensemble des tests inclus dans la suite et de préciser leur ordre d'exécution.

Pour créer une suite, il suffit de créer une classe de type TestSuite et d'appeler la méthode addTest() pour chaque classe de test à ajouter. Celle ci attend en paramètre une instance de la classe de tests qui sera ajoutée à la suite. L'objet de type TestSuite ainsi créé doit être renvoyé par une méthode dont la signature doit obligatoirement être public static Test suite(). Celle ci sera appellée par introspection par le TestRunner.

Il peut être pratique de définir une méthode main() dans la classe qui encapsule la suite de tests pour pouvoir exécuter le TestRunner de la console en exécutant directement la méthode statique Run(). Ceci évite de lancer JUnit sur la ligne de commandes.

Exemple :

```
import junit.framework.*;

public class ExecuterLesTests {

  public static Test suite() {
    TestSuite suite = new TestSuite("Tous les tests");
    suite.addTestSuite(MaClasseTest.class);
    suite.addTestSuite(MaClasse2Test.class);

    return suite;
  }

  public static void main(String args[]) {
    junit.textui.TestRunner.run(suite());
  }
}
```

Deux versions surchargées des constructeurs permettent de donner un nom à la suite de tests.

Un constructeur de la classe TestSuite permet de créer automatiquement par instrospection une suite de tests contenant tous les tests de la classe fournie en paramètre.

Exemple :

```
import junit.framework.*;

public class ExecuterLesTests2 {

  public static Test suiteDeTests() {
    TestSuite suite = new TestSuite(MaClasseTest.class,"Tous les tests");
    return suite;
  }

  public static void main(String args[]) {
    junit.textui.TestRunner.run(suiteDeTests());
  }
}
```

53.1.5. L'automatisation des tests avec Ant

L'automatisation des tests fait par JUnit au moment de la génération de l'application est particulièrement pratique. Ainsi Ant propose une tâche optionnelle particulière nommée junit pour exécuter un TestRunner dans la console.

Pour pouvoir utiliser cette tache, les fichiers junit.jar (fourni avec JUnit) et optional.jar (fourni avec Ant) doivent être accessibles dans le CLASSSPATH.

Cette tâche possède plusieurs attributs dont aucun n'est obligatoire et les principaux sont :

Attibut	Rôle	Valeur par défaut
printsummary	affiche un résumé statistique de l'exécution de chaque test	off
fork	exécution du TestRunner dans un JVM séparée	off
haltonerror	arrêt de la génération en cas d'erreur	off
haltonfailure	arrêt de la génération en cas d'échec d'un test	off
outfile	base du nom du fichier qui va contenir les résultats de l'exécution	

La tache <junit> peut avoir les élément fils suivants : <jvmarg>, <sysproperty>, <env>, <formatter>, <test>, <batchtest>

L'élément <formatter> permet de préciser le format de sortie des résultats de l'exécution des tests. Il possède l' attribut type qui précise le format (valeurs : xml, plain ou brief) et l'attribut usefile qui précise si les résultats doivent être envoyés dans un fichier (valeurs : true ou false)

L'élément <test> permet de préciser un cas de test simple ou une suite de test selon le contenu de la classe précisée par l'attribut name. Cet élément possède de nombreux attributs et il est possible d'utiliser un élément fils de type <formatter> pour définir le format de sortie du test.

L'élément <batchtest> permet de réaliser toute une série de tests. Cet élément possède de nombreux attributs et il est possible d'utiliser un élément fils de type <formatter> pour définir le format de sortie des tests. Les différentes classes dont les tests sont à exécuter sont précisées par un élément fils <fileset>.

La tache <junit> doit être exécutée après la compilation des classes à tester.

Exemple : extrait d'un fichier build.xml pour Ant

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<project name="TestAnt1" default="all">
   <description>Génération de l'application</description>
     <property name="bin" location="bin"/>
        <property name="src" location="src"/>
        <property name="build" location="build"/>
        <property name="doc" location="${build}/doc"/>
        <property name="lib" location="${build}/lib"/>

        <property name="junit_path" value="junit.jar"/>

        ...
        <target name="test" depends="compil" description="Executer les tests avec JUnit">
           <junit fork="yes" haltonerror="true" haltonfailure="on" printsummary="on">
              <formatter type="plain" usefile="false" />
              <test name="ExecuterLesTests"/>
              <classpath>
                 <pathelement location="${bin}"/>
                 <pathelement location="${junit_path}"/>
              </classpath>
           </junit>
        </target>
        ...

</project>
```

Cet exemple exécute les tests de la suite de tests encapsulés dans la classe ExecuterLesTests

53.1.6. Les extensions de JUnit

Junit est utilisé dans un certain nombre de projets qui proposent d'étendre les fonctionnalités de JUnit :

- Cactus : un framework open source de tests pour des composants serveur
- JunitReport : une tache Ant pour générer un rapport des tests effectués avec JUnit sous Ant
- JWebUnit : un framework open source de tests pour des applications web
- StrutsTestCase : extension de JUnit pour les tests d'application utilisant Struts 1.0.2 et 1.1
- XMLUnit : extension de JUnit pour les tests sur des documents XML

53.2. Cactus

Cactus est un framework open source de tests pour des composants serveur. Il est développé en s'appuyant sur JUnit par le projet Jakarta du groupe Apache.

La dernière version de Cactus peut être téléchargée sur le site http://jakarta.apache.org/cactus/.

54. Des bibliothèques open source

54.1. JFreeChart

JFreeChart est une bibliothèque open source qui permet de créer des données statistiques sous la forme de graphiques. Elle possède plusieurs formats dont le camembert, les barres ou les lignes et propose de nombreuses options de configuration pour personnaliser le rendu des graphiques. Elle peut s'utiliser dans des applications standalone ou des applications web et permet également d'exporter le graphique sous la forme d'une image.

http://www.jfree.org/jfreechart/

La version utilisée dans cette section est la 0.9.18.

Pour l'utiliser, il faut télécharger le fichier jfreechart−0.9.18.zip et le décompresser. Son utilisation nécessite l'ajout dans le classpath des fichiers jfreechart−0.9.18.zip et des fichiers .jar présents dans le répertoire lib décompressé.

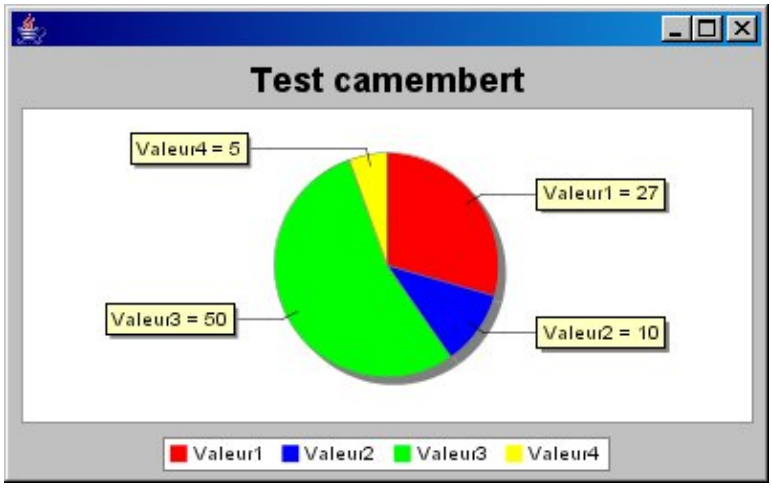
Les données utilisées dans le graphique sont encapsulées dans un objet de type Dataset. Il existe plusieurs sous type de cette classe en fonction du type de graphique souhaité.

Un objet de type JFreechart encapsule le graphique. Une instance d'un tel objet est obtenue en utilisant une des méthodes de la classe ChartFactory.

Un exemple avec un graphique en forme de camembert

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.*;
import org.jfree.data.*;

public class TestPieChart extends JFrame {
  private JPanel pnl;

  public TestPieChart() {
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(0);
      }
    });
    pnl = new JPanel(new BorderLayout());
    setContentPane(pnl);
    setSize(400, 250);

    DefaultPieDataset pieDataset = new DefaultPieDataset();
    pieDataset.setValue("Valeur1", new Integer(27));
    pieDataset.setValue("Valeur2", new Integer(10));
    pieDataset.setValue("Valeur3", new Integer(50));
    pieDataset.setValue("Valeur4", new Integer(5));

    JFreeChart pieChart = ChartFactory.createPieChart("Test camembert",
      pieDataset, true, true, true);
    ChartPanel cPanel = new ChartPanel(pieChart);
    pnl.add(cPanel);
```

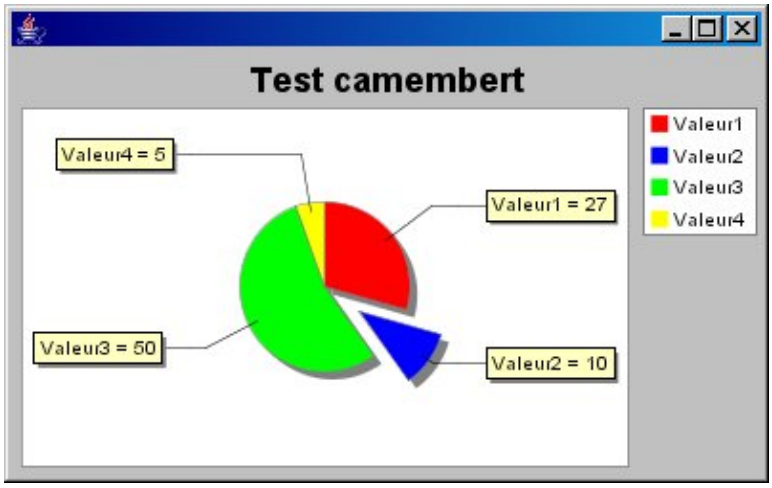
```
  }

  public static void main(String args[]) {
    TestPieChart tpc = new TestPieChart();
    tpc.setVisible(true);
  }
}
```



Pour chaque graphique, il existe de nombreuses possibilités de configuration.

Exemple :

```
...
    JFreeChart pieChart = ChartFactory.createPieChart("Test camembert",
      pieDataset, true, true, true);
    PiePlot piePlot = (PiePlot) pieChart.getPlot();
    piePlot.setExplodePercent(1, 0.5);
    Legend legend = pieChart.getLegend();
    legend.setAnchor(Legend.EAST_NORTHEAST);
    ChartPanel cPanel = new ChartPanel(pieChart);
...
```



Il est très facile d'exporter le graphique dans un flux.

Exemple : enregistrement du graphique dans un fichier<

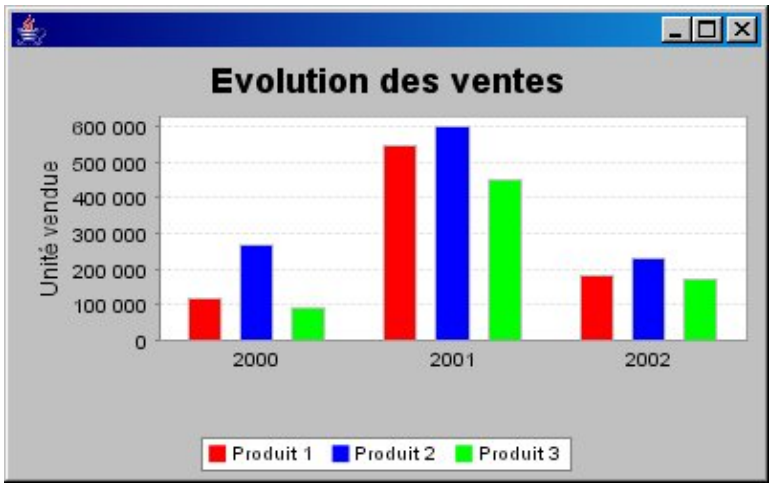
```
...
    File fichier = new File("image.png");
    try {
      ChartUtilities.saveChartAsPNG(fichier, pieChart, 400, 250);
    } catch (IOException e) {
```

```
      e.printStackTrace();
    }
...
```

JFreeChart propose aussi plusieurs autres types de graphiques dont les graphiques en forme de barres.

Exemple : un graphique sous formes de barres

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.*;
import org.jfree.data.*;

public class TestBarChart extends JFrame {
  private JPanel pnl;

  public TestBarChart() {
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(0);
      }
    });
    pnl = new JPanel(new BorderLayout());
    setContentPane(pnl);
    setSize(400, 250);

    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    dataset.addValue(120000.0, "Produit 1", "2000");
    dataset.addValue(550000.0, "Produit 1", "2001");
    dataset.addValue(180000.0, "Produit 1", "2002");
    dataset.addValue(270000.0, "Produit 2", "2000");
    dataset.addValue(600000.0, "Produit 2", "2001");
    dataset.addValue(230000.0, "Produit 2", "2002");
    dataset.addValue(90000.0, "Produit 3", "2000");
    dataset.addValue(450000.0, "Produit 3", "2001");
    dataset.addValue(170000.0, "Produit 3", "2002");

    JFreeChart barChart = ChartFactory.createBarChart("Evolution des ventes", "",
      "Unité vendue", dataset, PlotOrientation.VERTICAL, true, true, false);
    ChartPanel cPanel = new ChartPanel(barChart);
    pnl.add(cPanel);
  }

  public static void main(String[] args) {
    TestBarChart tbc = new TestBarChart();
    tbc.setVisible(true);
  }
}
```



JFreechart peut aussi être mis en oeuvre dans une application web, le plus pratique étant d'utiliser une servlet qui renvoie dans la réponse une image générée par JfreeChart.

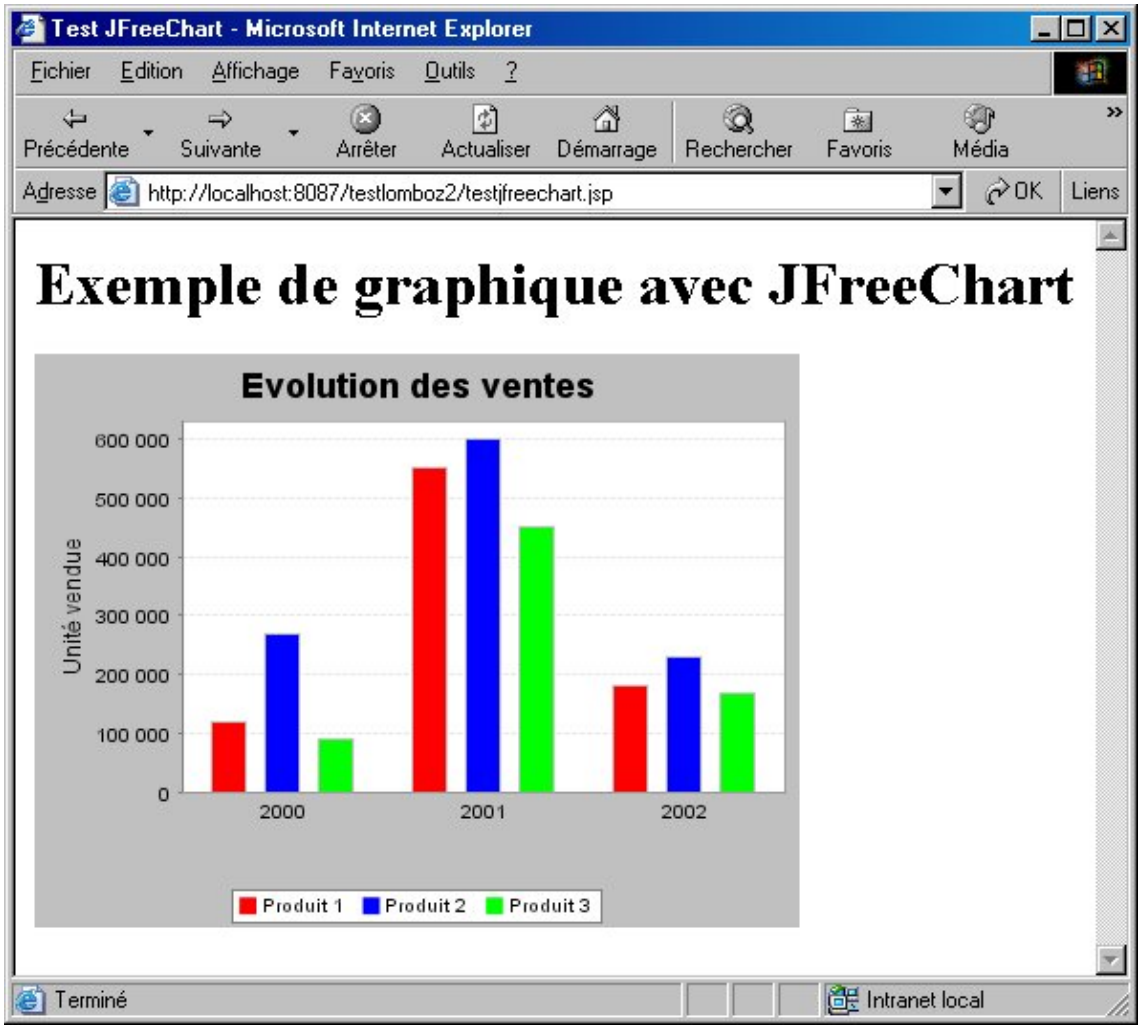
Exemple : JSP qui affiche le graphique

```
<%@ page language="java" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Test JFreeChart</title>
</head>
<body bgcolor="#FFFFFF">
<H1>Exemple de graphique avec JFreeChart</h1>
<img src="/testlomboz2/ServletBarChart">
</body>
</html>
```

Dans l'exemple précédent, l'image contenant le graphique est généré par une sevlet.

Exemple : servlet qui générère l'image

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.*;
import org.jfree.data.*;

public class ServletBarChart extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    dataset.addValue(120000.0, "Produit 1", "2000");
    dataset.addValue(550000.0, "Produit 1", "2001");
    dataset.addValue(180000.0, "Produit 1", "2002");
    dataset.addValue(270000.0, "Produit 2", "2000");
    dataset.addValue(600000.0, "Produit 2", "2001");
    dataset.addValue(230000.0, "Produit 2", "2002");
    dataset.addValue(90000.0, "Produit 3", "2000");
    dataset.addValue(450000.0, "Produit 3", "2001");
    dataset.addValue(170000.0, "Produit 3", "2002");

    JFreeChart barChart = ChartFactory.createBarChart("Evolution des ventes", "",
      "Unité vendue", dataset, PlotOrientation.VERTICAL, true, true, false);
    OutputStream out = response.getOutputStream();
    response.setContentType("image/png");
    ChartUtilities.writeChartAsPNG(out, barChart, 400, 300);
  }
}
```



Cette section n'a proposé qu'une introduction à JFreeChart en proposant quelques exemples très simple sur les nombreuses possibilités de cette puissante bibliothèque.

55. Des outils open source pour faciliter le développement

La communauté open source propose de nombreuses bibliothèques mais aussi des outils dont le but est de faciliter le travail des développeurs. Certains de ces outils sont détaillés dans des chapitres dédiés notamment Ant, Maven et JUnit. Ce chapitre va présenter d'autres outils open source pouvant être regroupés dans plusieurs catégories : contrôle de la qualité des sources et génération et mis en forme de code.

La génération de certains morceaux de code ou de fichiers de configuration peut parfois être fastidieuse voir même répétitif dans certains cas. Pour faciliter le travail des développeurs, des outils open source ont été développés par la communauté afin de générer certains morceaux de code. Ce chapitre présente deux outils open source : XDoclet et Middlegen.

La qualité du code source est un facteur important pour tous développements. Ainsi certains outils permettent de faire des vérifications sur des règles de codification dans le code source. C'est le cas pour l'outil CheckStyle.

Ce chapitre contient plusieurs sections :

- CheckStyle
- Jalopy
- XDoclet
- Middlegen

55.1. CheckStyle

CheckStyle est un outil open source qui propose de puissantes fonctionnalités pour appliquer des contrôles sur le respect de règles de codifications.

Pour définir les contrôles réalisés lors de son exécution CheckStyle utilise une configuration qui repose sur des modules. Cette configuration est définie dans un fichier XML qui précise les modules utilisés et pour chacun d'entre eux leurs paramètres.

Le plus simple est d'utiliser le fichier de configuration nommé sun_checks.xml fourni avec CheckStyle. Cette configuration propose d'effectuer des contrôles de respect des normes de codification proposée par Sun. Il est aussi possible de définir son propre fichier de configuration.

Le site officiel de CheckStyle est à l'URL : http://checkstyle.sourceforge.net/

La version utilisée dans cette section est la 3.4

55.1.1. Installation

Il faut télécharger le fichier checkstyle−3.4.zip sur le site de ChekStyle et le décompresser dans un répertoire du système.

La décompression de ce fichier créé un répertoire checkstyle−3.4 contenant lui même les bibliothèques utiles et deux répertoires (docs et contrib).

CheckStyle peut s'utiliser de deux façons :

- en ligne de commande
- comme une tâche Ant ce qui permet d'automatiser son exécution

55.1.2. Utilisation avec Ant

Pour utiliser CheckStyle, le plus simple est d'ajouter la bibliothèque checkstyle−all−3.4.jar au classpath.

Dans les exemples de cette section, la structure de répertoires suivante est utilisée :

/bin
/lib
/outils
/outils/lib
/outils/checkstyle
/src
/temp
/temp/checkstyle

Le fichier checkstyle−all−3.4.jar est copié dans le répertoire outils/lib et le fichier sun_checks.xml fourni par CheckStyle est copié dans le répertoire outils/checkstyle.

Il faut déclarer le tag CheckStyle dans Ant en utilisant le tag <taskdef> et définir une tâche qui va utiliser le tag <CheckStyle>.

Exemple : fichier source de test

```
public class MaClasse {

  public static void main() {
    System.out.println("Bonjour");
  }

}
```

Le fichier de build ci−dessous sera exécuté par Ant.

Exemple :

```
<project name="utilisation de checkstyle" default="compile" basedir=".">
  <!-- Definition des proprietes du projet -->
  <property name="projet.sources.dir"       value="src"/>
  <property name="projet.bin.dir"           value="bin"/>
  <property name="projet.lib.dir"           value="lib"/>
  <property name="projet.temp.dir"          value="temp"/>
  <property name="projet.outils.dir"        value="outils"/>
  <property name="projet.outils.lib.dir"    value="${projet.outils.dir}/lib"/>

  <!-- Definition du classpath du projet -->
  <path id="projet.classpath">
    <fileset dir="${projet.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <fileset dir="${projet.outils.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${projet.bin.dir}" />
  </path>

  <!-- Declaration de la tache Ant permettant l'execution de checkstyle -->
  <taskdef resource="checkstyletask.properties"
    classpathref="projet.classpath" />
```

```
  <!-- execution de checkstyle -->
  <target name="checkstyle" description="CheckStyle">
    <checkstyle config="outils/checkstyle/sun_checks.xml">
      <fileset dir="${projet.sources.dir}" includes="**/*.java"/>
      <formatter type="plain"/>
    </checkstyle>
  </target>

  <!-- Compilation des classes du projet -->
  <target name="compile" depends="checkstyle" description="Compilation des classes">
    <javac srcdir="${projet.sources.dir}"
      destdir="${projet.bin.dir}"
      debug="on"
      optimize="off"
      deprecation="on">
      <classpath refid="projet.classpath"/>
    </javac>
  </target>

</project>
```

Résultat :

```
C:\java\test\testcheckstyle>ant
Buildfile: build.xml
checkstyle:
[checkstyle] C:\java\test\testcheckstyle\src\package.html:0: Missing package doc
umentation file.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:0: File does not end
with a newline.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:2: Missing a Javadoc
comment.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:2:1: Utility classes
should not have a public or default constructor.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:4:3: Missing a Javado
c comment.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:5: Line has trailing
spaces.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:5:35: La ligne contie
nt un caractÞre tabulation.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:7: Line has trailing
spaces.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:7:1: La ligne contien
t un caractÞre tabulation.
BUILD FAILED
file:C:/java/test/testcheckstyle/build.xml:27: Got 9 errors.
      Total time: 7 seconds
```

Le tag <checkstyle> possède plusieurs attributs :

Nom	Rôle
file	précise le nom de l'unique fichier à vérifier. Pour préciser un ensemble de fichiers, il faut définir un ensemble de fichiers grâce à un tag fils <fileset>
config	précise le nom du fichier de configuration des modules de ChecksStyle
failOnViolation	précise si les traitements de Ant doivent être stoppés en cas d'échec des contrôles. La valeur par défaut est true
failureProperty	précise le nom d'une propriété qui sera valorisée en cas d'échec des contrôles

Exemple :

```
...
  <!-- execution de checkstyle -->
  <target name="checkstyle" description="CheckStyle">
    <checkstyle config="outils/checkstyle/sun_checks.xml" failOnViolation="false">
      <fileset dir="${projet.sources.dir}" includes="**/*.java"/>
```

```
      <formatter type="plain"/>
    </checkstyle>
  </target>
...
```

Résultat :

```
C:\java\test\testcheckstyle>ant
Buildfile: build.xml
checkstyle:
[checkstyle] C:\java\test\testcheckstyle\src\package.html:0: Missing package doc
umentation file.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:2: Missing a Javadoc
comment.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:2:1: Utility classes
should not have a public or default constructor.
[checkstyle] C:\java\test\testcheckstyle\src\MaClasse.java:4:3: Missing a Javado
c comment.
compile:
[javac] Compiling 1 source file to C:\java\test\testcheckstyle\bin
BUILD SUCCESSFUL
      Total time: 11 seconds
```

Il est possible de préciser deux types de format de sortie des résultats lors de l'exécution de CheckStyle. Le format de sortie des résultats est précisé par un tag fils <formatter>. Ce tag possède deux attributs :

Nom	Rôle
type	précise le type. Deux valeurs sont possibles : plain (par défaut) et xml
toFile	précise un fichier qui va contenir les résultats dont le format correspondra au type (par défaut la sortie standard de la console)

Exemple :

```
...
  <!-- execution de checkstyle -->
  <target name="checkstyle" description="CheckStyle">
    <checkstyle config="outils/checkstyle/sun_checks.xml" failOnViolation="false">
      <fileset dir="${projet.sources.dir}" includes="**/*.java"/>
      <formatter type="xml" toFile="${projet.temp.dir}/checkstyle_erreurs.xml"/>
    </checkstyle>
  </target>
...
```

Il est alors possible d'appliquer une feuille de style sur le fichier XML généré afin de créer un rapport dans un format dédié. L'exemple suivant utilise une feuille de style fournie par CheckStyle dans le répertoire contrib : cette feuille, nommée checkstyle-frames.xsl, est copiée dans le répertoire outils/checkstyle.

Exemple :

```
...
  <!-- execution de checkstyle -->
  <target name="checkstyle" description="CheckStyle">
    <checkstyle config="${projet.outils.dir}/checkstyle/sun_checks.xml" failOnViolation="false">
      <fileset dir="${projet.sources.dir}" includes="**/*.java"/>
      <formatter type="xml" toFile="${projet.temp.dir}/checkstyle/checkstyle_erreurs.xml"/>
    </checkstyle>
    <style in="${projet.temp.dir}/checkstyle/checkstyle_erreurs.xml"
      out="${projet.temp.dir}/checkstyle/checkstyle_rapport.htm"
      style="${projet.outils.dir}/checkstyle/checkstyle-frames.xsl"/>
  </target>
...
```

Exemple :

```
C:\java\test\testcheckstyle>ant
Buildfile: build.xml
checkstyle:
[style] Processing C:\java\test\testcheckstyle\temp\checkstyle\checkstyle_er
reurs.xml to C:\java\test\testcheckstyle\temp\checkstyle\checkstyle_rapport.htm
[style] Loading stylesheet C:\java\test\testcheckstyle\outils\checkstyle\che
ckstyle-frames.xsl
compile:
BUILD SUCCESSFUL
Total time: 8 seconds
```

Il est possible d'utiliser d'autres feuilles de style fournies par CheckStyle ou de définir sa propre feuille de style.

55.1.3. Utilisation en ligne de commandes

Pour utiliser CheckStyle en ligne de commandes, il faut ajouter le fichier checkstyle-all-3.4.jar au classpath par exemple en utilisant l'option –cp de l'interpréteur Java.

La classe à exécuter est com.puppycrawl.tools.checkstyle.Main

CheckStyle accepte plusieurs paramètres pour son exécution :

Option	Rôle
-c fichier_de_configuration	précise le fichier de configuration
-f format	précise le format (plain ou xml)
-o fichier	précise le fichier qui va contenir les résultats
-r	précise le répertoire dont les fichiers sources vont être récursivement traités

Exemple :

```
java -cp outils\lib\checkstyle-all-3.4.jar com.puppycrawl.tools.checkstyle.Main
  -c outils\checkstyle/sun_checks.xml -r src
```

Exemple :

```
...
C:\java\test\testcheckstyle>java -cp outils\lib\checkstyle-all-3.4.jar com.puppy
crawl.tools.checkstyle.Main -c outils\checkstyle/sun_checks.xml -r src
Starting audit...
C:\java\test\testcheckstyle\src\package.html:0: Missing package documentation fi
le.
src\MaClasse.java.bak:0: File does not end with a newline.
src\MaClasse.java:2: Missing a Javadoc comment.
src\MaClasse.java:2:1: Utility classes should not have a public or default const
ructor.
src\MaClasse.java:4:3: Missing a Javadoc comment.
Audit done.
...
```

55.2. Jalopy

Jalopy est un outil open source qui propose de formater les fichiers source selon des règles définies.

Le site web officiel de Jalopy est http://jalopy.sourceforge.net

Jalopy propose entre autre les fonctionnalités suivantes :

- formatage des accolades selon plusieurs format (C, Sun, GNU)
- indentation du code
- gestion des sauts de ligne
- génération automatique ou vérification des commentaires Javadoc
- ordonnancement des éléments qui composent la classe
- ajout de texte au début et la fin de chaque fichier
- des plug in pour une intégration dans plusieurs outils : ant, Eclipse, Jbuilder, ...
- ...

Pour connaître les règles de formatage à appliquer, Jalopy utilise une convention qui est un ensemble de paramètres.

Jalopy peut être utilisé grâce à ces plug ins de plusieurs façons notamment avec Ant, en ligne commande ou avec certains IDE.

La version de Jalopy utilisée dans cette section est la 1.0.B10

55.2.1. Utilisation avec Ant

Le plus simple est d'utiliser Jalopy avec Ant pour automatiser son utilisation : pour cela, il faut télécharger le fichier jalopy−ant−0.6.2.zip et le décompresser dans un répertoire du système.

La structure de l'arborescence du projet utilisé dans cette section est la suivante :

/bin
/lib
/outils
/outils/lib
/src

Le répertoire src contient les sources Java à formater.

Les fichiers du répertoires lib de Jalopy sont copiés dans le répertoire outils/lib du projet.

Exemple : le fichier source qui sera formaté

```
public class MaClasse {public static void main() { System.out.println("Bonjour"); }}
```

Il faut définir un fichier build.xml pour Ant qui va contenir les différentes tâches du projet dont une permettant l'appel à Jalopy.

Exemple :

```
<project name="utilisation de jalopy" default="jalopy" basedir=".">
  <!−− Definition des proprietes du projet −−>
  <property name="projet.sources.dir" value="src"/>
  <property name="projet.bin.dir" value="bin"/>
  <property name="projet.lib.dir" value="lib"/>
  <property name="projet.temp.dir" value="temp"/>
  <property name="projet.outils.dir" value="outils"/>
  <property name="projet.outils.lib.dir" value="${projet.outils.dir}/lib"/>

  <!−− Definition du classpath du projet −−>
  <path id="projet.classpath">
    <fileset dir="${projet.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <fileset dir="${projet.outils.lib.dir}">
```

```
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${projet.bin.dir}" />
  </path>

  <!-- Declaration de la tache Ant permettant l'execution de jalopy -->
  <taskdef name="jalopy"
         classname="de.hunsicker.jalopy.plugin.ant.AntPlugin"
      classpathref="projet.classpath" />

  <!-- execution de jalopy -->
  <target name="jalopy" description="Jalopy" depends="compile" >
    <jalopy loglevel="info"
                threads="2"
           classpathref="projet.classpath">
      <fileset dir="${projet.sources.dir}">
        <include name="**/*.java" />
      </fileset>
    </jalopy>
  </target>

  <!-- Compilation des classes du projet -->
  <target name="compile" description="Compilation des classes">
    <javac srcdir="${projet.sources.dir}"
             destdir="${projet.bin.dir}"
               debug="on"
            optimize="off"
         deprecation="on">
      <classpath refid="projet.classpath"/>
    </javac>
  </target>

</project>
```

Exemple :

```
C:\java\test\testjalopy>ant
Buildfile: build.xml

compile:
    [javac] Compiling 1 source file to C:\java\test\testjalopy\bin

jalopy:
   [jalopy] Jalopy Java Source Code Formatter 1.0b10
   [jalopy] Format 1 source file
   [jalopy] C:\java\test\testjalopy\src\MaClasse.java:0:0: Parse
   [jalopy] 1 source file formatted

BUILD SUCCESSFUL
Total time: 9 seconds
```

Suite à l'exécution de Jalopy, le code du fichier est reformaté.

Exemple :

```
public class MaClasse {
    public static void main() {
        System.out.println("Bonjour");
    }
}
```

Il est fortement recommandé de réaliser la tâche de compilation des sources avant leur formatage car pour assurer un formatage correct les sources doivent être corrects syntaxiquement parlant.

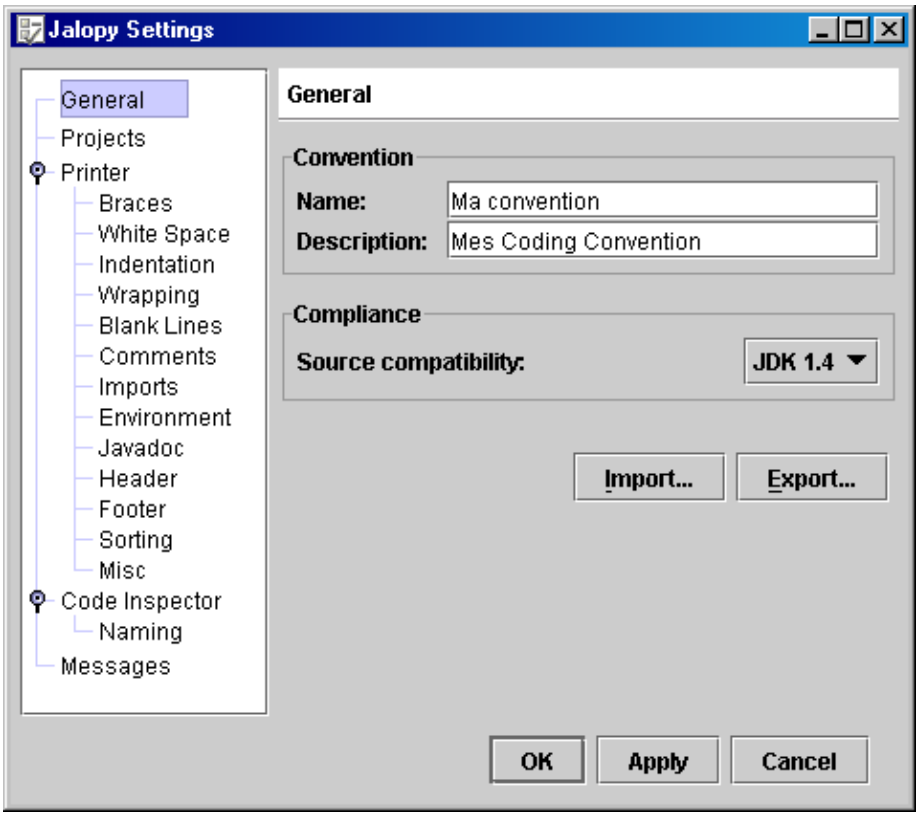
55.2.2. Les conventions

Jalopy est hautement paramétrable. Les options utilisées sont regroupées dans une convention.

Tous ses paramètres sont stockés dans le sous répertoire .jalopy du répertoire Home de l'utilisateur.

Pour faciliter la gestion de ces paramètres, Jalopy propose un outil graphique qui permet de gérer les conventions.

Pour exécuter cet outil, il suffit de lancer le script preferences dans le répertoire bin de Jalopy (preferences.bat sous Windows et preferences.sh sous Unix).



Toutes les nombreuses options de formatage d'une convention peuvent être réglées via cet outil. Consultez la documentation fournie avec Jalopy pour un détail de chaque option.

Une fonctionnalité particulièrement utile de cet outil est de proposer une pré−visualisation d'un exemple mettant en oeuvre les options sélectionnées.

Voici un exemple avec quelques personnalisations notamment une gestion des clauses import, la génération des commentaires Javadoc :

Exemple :

```
import java.util.*;

public class MaClasse {
    public static void main() {
        List liste = new ArrayList();
        System.out.println("Bonjour");
    }

    /**
     *
     */
    private int maMethode(int valeur) {
        return valeur * 2;
    }
}
```

Résultat :

```
C:\java\test\testjalopy>ant
Buildfile: build.xml

compile:
    [javac] Compiling 1 source file to C:\java\test\testjalopy\bin

jalopy:
   [jalopy] Jalopy Java Source Code Formatter 1.0b10
   [jalopy] Format 1 source file
   [jalopy] C:\java\test\testjalopy\src\MaClasse.java:0:0: Parse
   [jalopy] C:\java\test\testjalopy\src\MaClasse.java:1:0: On-demand import "jav
a.util.List" expanded
   [jalopy] C:\java\test\testjalopy\src\MaClasse.java:1:0: On-demand import "jav
a.util.ArrayList" expanded
   [jalopy] C:\java\test\testjalopy\src\MaClasse.java:11:1: Generated Javadoc co
mment
   [jalopy] C:\java\test\testjalopy\src\MaClasse.java:18:3: Generated Javadoc co
mment
   [jalopy] 1 source file formatted

BUILD SUCCESSFUL
Total time: 10 seconds
```

Voici le source du code reformaté :

Exemple :

```
//============================================================================
// fichier :       MaClasse.java
// projet :        $project$
//
// Modification :  date :       $Date$
//                 auteur :     $Author$
//                 revision :   $Revision$
//----------------------------------------------------------------------------
// copyright:   JMD
//============================================================================

import java.util.ArrayList;
import java.util.List;


/**
 * DOCUMENT ME!
 *
 * @author $author$
 * @version $Revision$
 */
public class MaClasse {
  /**
   * DOCUMENT ME!
   */
  public static void main() {
    List liste = new ArrayList();
    System.out.println("Bonjour");
    System.out.println("");
  }

  /**
   * Calculer le double
   *
   * @param valeur DOCUMENT ME!
   *
   * @return DOCUMENT ME!
   */
  private int maMethode(int valeur) {
    return valeur * 2;
  }
}
```

```
// fin du fichier
```

55.3. XDoclet

La suite de ce chapitre sera développée dans une version future de ce document

55.4. Middlegen

La suite de ce chapitre sera développée dans une version future de ce document

Partie 6 : Développement d'applications mobiles

Le marché des machines portables est en pleine expansion : téléphones mobiles, PDA, ... De plus en plus d'applications s'exécutent sur des machines embarquées.

Sun propose une édition particulière de Java pour ce type de développement : J2ME (Java 2 Micro Edition).

Cette partie contient les chapitres suivants :

- J2ME : présente la plate−forme java pour le développement d'applications sur des appareils mobiles tel que des PDA ou des téléphones cellulaires
- CLDC : présente les packages et les classes de la configuration CLDC
- MIDP : propose une présentation et une mise en oeuvre du profil MIDP pour le développement d'applications mobiles
- CDC : présente les packages et les classes de la configuration CDC
- Les profils du CDC : propose une présentation et une mise en oeuvre des profils pouvant être mis en oeuvre avec la CDC
- Les autres technologies pour les applications mobiles : propose une présentation des autres technologies basées sur Java pour développer des applications mobiles

56. J2ME

J2ME est la plate−forme java pour développer des applications sur des appareils mobiles tel que des PDA, des téléphones cellulaires, des terminaux de points de vente, des systèmes de navigations pour voiture, ...

C'est une sorte de retour aux sources puisque Java avait été initialement développé pour piloter des appareils électroniques.

Ce chapitre contient plusieurs sections :

- Présentation de J2ME : présentation rapide de J2ME
- Les configurations : présentation des deux configurations sur lesquels la plate−forme J2ME repose
- Les profiles : présentation des profiles qui enrichissent les configurations pour un type machines ou à une fonctionnalité spécifique
- J2ME Wireless Toolkit 1.0.4 : Installation et mise en oeuvre de cet outil proposé par Sun pour le développement d'applications utilisant MIDP 1.0
- J2ME wireless toolkit 2.1 : Installation et mise en oeuvre de cet outil proposé par Sun pour le développement d'applications utilisant MIDP 1.0 et 2.0

56.1. Présentation de J2ME

Historiquement, Sun a proposé plusieurs plate−formes pour le développement d'applications sur des machines possédant des ressources réduites, typiquement celles ne pouvant exécuter une JVM répondant aux spécifications complètes de la plate−forme J2SE.

- JavaCard : pour le développement sur des cartes à puces
- EmbeddedJava :
- PersonnalJava : pour le développement sur des machines possédant au moins 2mo de mémoire

En 1999, Sun propose de mieux structurer ces différentes plate−formes sous l'appellation J2ME (Java 2 Micro Edition). Seule le plate−forme JavaCard n'est pas incluse dans J2ME et reste à part.

Par rapport à J2SE, J2ME utilise des machines virtuelles différentes. Certaines classes de base de l'API sont communes avec cependant de nombreuses omissions dans l'API J2ME.

L'ensemble des appareils sur lequel peut s'exécuter une application écrite avec J2ME est tellement vaste et disparate que J2ME est composé de plusieurs parties : les configurations et les profiles qui sont spécifiés par le JCP. J2ME propose donc une architecture modulaire.

Chaque configuration peut être utilisée avec un ensemble de packages optionnels qui permet d'utiliser des technologies particulières (Bluetooth, services web, lecteur de codes barre, etc ...). Ces packages sont le plus souvent dépendant du matériel.

L'inconvénient de ce principe est qu'il déroge à la devise de Java "Write Once, Run Anywhere". Ceci reste cependant partiellement vrai pour des applications développées pour un profile praticulier. Il ne faut cependant pas oublier que les types de machines cibles de J2ME sont tellement differents (du téléphone mobile au set top box), qu'il est surement impossible de trouver un dénominateur commun. Ceci associé à l'explosion du marché des machines mobiles explique

les nombreuses évolutions en cours de la plate−forme.

J2ME est la plate−forme Java la plus récente.

De plus amples informations peuvent être obtenues sur les deux sites de Sun :

- http://wireless.java.sun.com/
- http://java.sun.com/j2me/

et sur les sites

- http://www.javamobiles.com/
- http://www.microjava.com/

56.2. Les configurations

Les configurations définissent les caractéristiques de bases d'un environnement d'exécution pour un certain type de machine possédant un ensemble de caractéristiques et de ressources similaires. Elles se composent d'une machine virtuelle et d'un ensemble d'API de base.

Deux configurations sont actuellement définies :

- CLDC (Connected Limited Device Configuration)
- CDC (Connected Device Configuration).

La CLDC 1.0 est spécifiée dans la JSR 030 : elle concerne des appareils possédant des ressources faibles (moins de 512 Kb de RAM, faible vitesse du processeur, connexion réseau limitée et intermittente) et une interface utilisateur réduite (par exemple un téléphone mobile ou un PDA "bas de gamme"). Elle s'utilise sur une machine virtuelle KVM. La version 1.1 est le résultat des spécifications de la JSR 139 : une des améliorations les plus importantes est le support des nombres flottants.

La CDC est spécifié dans la JSR 036 : elle concerne des appareils possédant des ressources plus importantes (au moins 2Mb de RAM, un processeur 32 bits, une meilleure connexion au réseau), par exemple un settop box ou certains PDA "haut de gamme". Elle s'utilise sur une machine virtuelle CVM

56.3. Les profiles

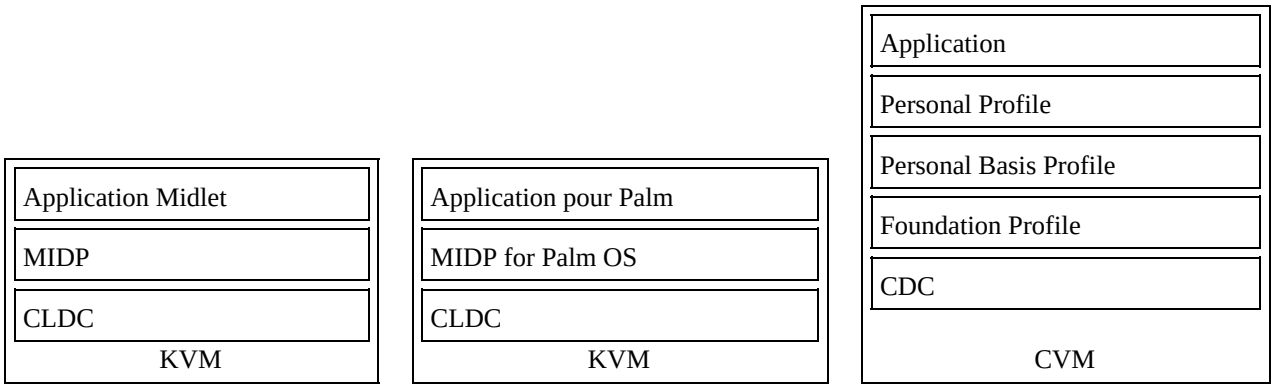
Les profiles se composent d'un ensemble d'API particulieres à un type de machines ou à une fonctionnalité spécifique. Ils permettent l'utilisation de fonctionnalités précises et doivent être associés à une configuration. Ils permettent donc d'assurer une certaine modularité à la plate−forme J2ME.

Il existe plusieurs profiles :

Profil	Configuration	JSR	
MIDP 1.0	CLDC	37	Package javax.microedition.*
Foundation Profile	CDC	46	
Personal Profile	CDC	62	
MIPD 2.0	CLDC	118	
Personal Basis Profile	CDC	129	
RMI optional Profile	CDC	66	
Mobile Media API (MMAPI) 1.1	CLDC	135	Permet la lecture de clips audio et vidéo
PDA		75	

JDBC optional Profile	CDC	169	
Wireless Messaging API (WMA) 1.1	CLDC	120	Permet l'envoi et la réception de SMS

Les utilisations possibles des profils sont :



MIDP est un profile standard qui n'est pas défini pour une machine particulière mais pour un ensemble de machines embarquées possédant des ressources et une interface graphique limitée.

Sun a développé un profil particulier nommé KJava pour le développement spécifique sur Palm. Ce profile a été remplacé par un nouveau profil nommé MIDP for Palm OS.

Le Foundation Profile est un profil de base qui s'utilise avec CDC. Ce profil ne permet pas de développer des IHM. Il faut lui associer un des deux profils suivants :

- le Personal Basic Profile permet le développement d'application connectée avec le réseau
- le Personal Profile est un profil qui permet le développement complet d'une IHM et d'applet grace à AWT.

PersonalJava est remplacé par le Personal Profile.

Le choix du ou des profils utilisés pour les développements est important car il conditionne l'exécution de l'application sur un type de machine supporté par le profil.

Cette multitude de profils peut engendrer un certain nombre de problème lors de l'exécution d'une application sur différents périphériques car il n'y a pas la certitude d'avoir à disposition les profils nécessaires. Pour résoudre ce problème, une spécification particulière issue des travaux de la JSR 185 et nommée Java Technology for the Wireless Industry (JTWI) a été développée. Cette spécification impose aux périphériques qui la respectent de mettre en oeuvre au minimum : CLDC 1.0, MIDP 2.0, Wireless Messaging API 1.1 et Mobile Media API 1.1. Son but est donc d'assumer une meilleure compatibilité entre les applications et les différents téléphones mobiles sur lesquelles elles s'exécutent.

56.4. J2ME Wireless Toolkit 1.0.4

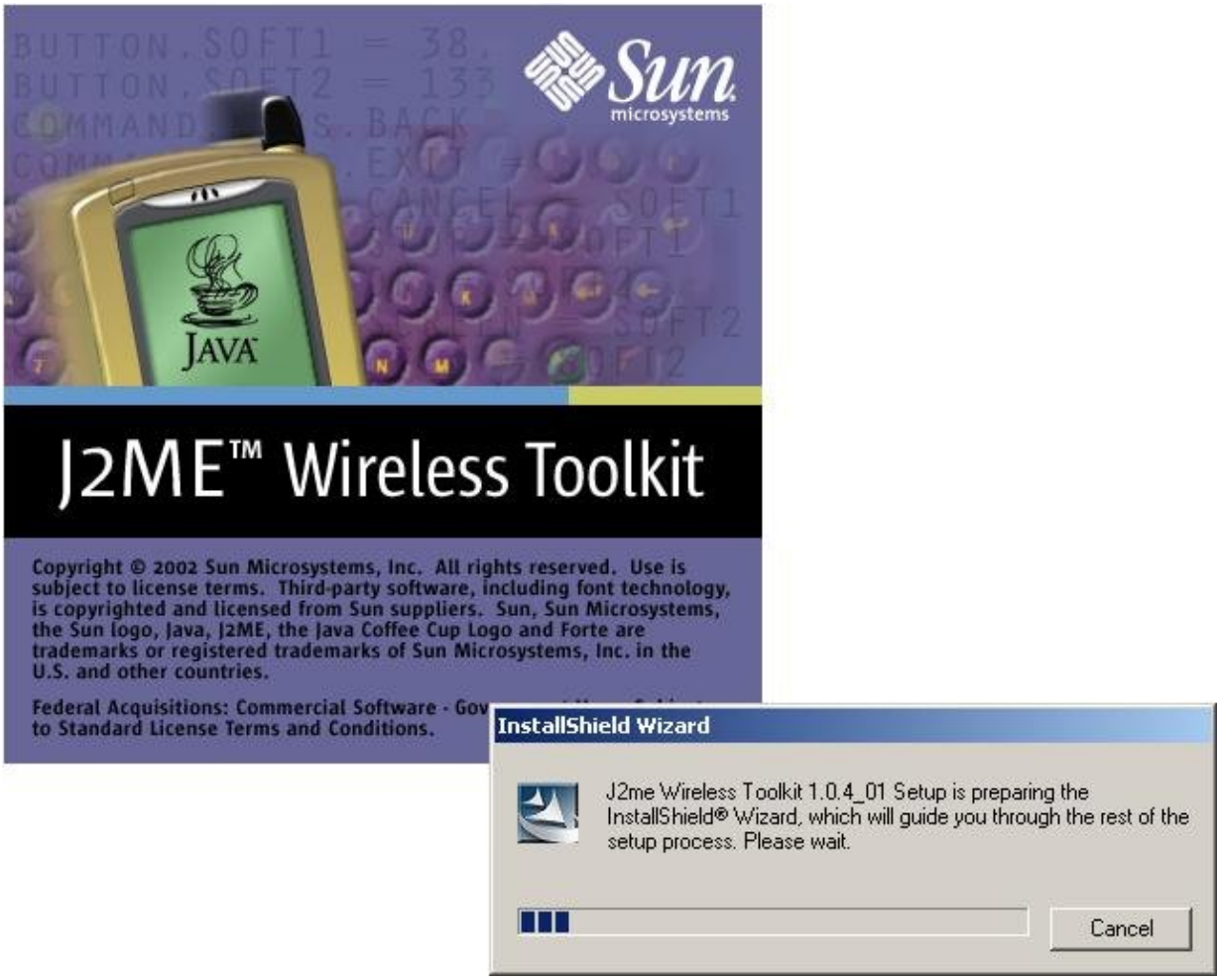
Sun propose un outil pour développer des applications J2ME utilisant CLDC/MIDP. Cet outil peut être téléchargé à l'url suivante : http://java.sun.com/products/j2mewtoolkit/index.html

La version 1.0.4 de cet outil permet de développer des applications utilisant MIDP 1.0.

56.4.1. Installation du J2ME Wireless Toolkit 1.0.4

L'installation ci dessous concerne la version 1.0.4.

Il faut exécuter le fichier j2me_wireless_toolkit−1_0_4_01−bin−win.exe

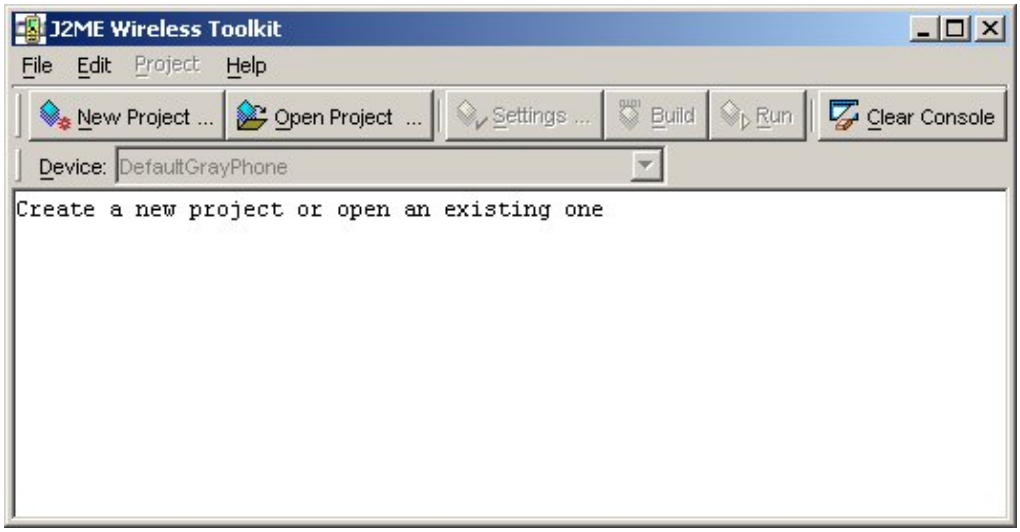


Il faut suivre les instructions suivantes, guidées par l'assistant d'installation :

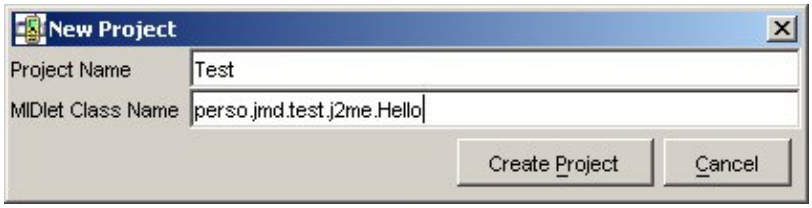
- sur la page d'accueil (welcome) , cliquez sur "Suivant"
- sur la page d'acception de la licence, lire la licence et l'approuver en cliquant sur "Yes"
- sur la page de sélection de localisation de la JVM, cliquez sur "Next" (sélectionner l'emplacement si aucune JVM n'a été détectée automatiquement)
- sélectionner l'emplacement de l'installation de l'outil et cliquez sur "Next"
- cliquez sur "Next" pour accepter le menu par défaut dans le menu "Démarrer/Programme"
- sur la page de résumé des opérations, cliquez sur "Next"
- sur le dernière page (Complete), cliquez sur "Finish"

56.4.2. Premiers pas

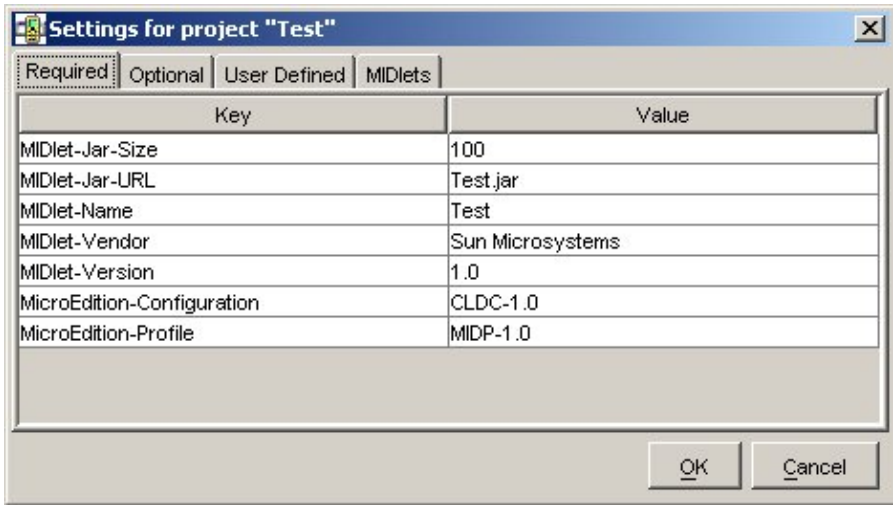
Il faut exécuter l'outil KToolBar.



Pour créer un projet, il faut cliquer sur le bouton "New Project" ou sur l'option "New Project" du menu "File".



Il faut saisir le nom du projet et le nom qualifié de la midlet puis cliquer sur "Create Project".



Il faut ensuite créer la ou les classes dans le répertoire src de l'arbrorescence du projet.

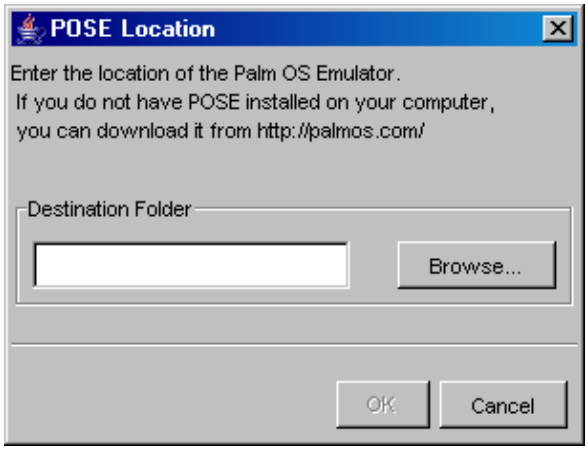
Pour construire le projet, il faut cliquer sur le bouton "Build".

Pour exécuter le projet, il suffit de choisir le type d'émulateur à utiliser et cliquer sur le bouton "Run".

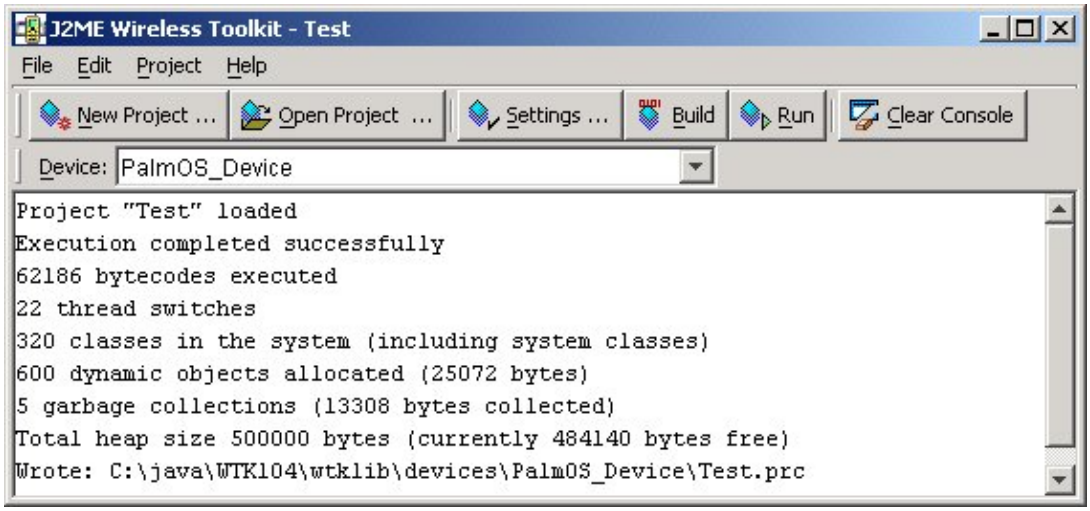
Exemple : avec l'émulateur de téléphone par défaut.

Cliquer sur l'application "Test" puis cliquer sur le bouton entre les flèches

Il est aussi possible d'utiliser l'émulateur Palm POSE (Palm O.S. Emulator). L'outil demande le chemin d'accès à POSE,



Puis l'outil génère un fichier .prc.



Enfin, il lance l'émulateur et installe le fichier pour l'exécuter.

Pour plus de détails, voir la section sur MIDP for Palm OS.

56.5. J2ME wireless toolkit 2.1

La version 2.0 permet d'utiliser MIDP 1.0 ou 2.0 ainsi que les API optionnels Mobile Media et Wireless Messaging . Il peut être intégré dans d'autres IDE tel que Sun Studio Mobile Edition ou JBuilder.

La version 2.1 permet d'utiliser CLDC 1.1 et l'API J2ME Web service et de développer des applications pour des périphériques qui respectent les spécifications JTWI.

56.5.1. Installation du J2ME Wireless Toolkit 2.1

La version 2.1 du J2ME Wireless Toolkit nécessite la présence sur le système d'un J2SE 1.4 minimum.

Elle permet le développement d'applications répondant aux spécifications de la JSR−185 (Java Technology for the Wireless Industry) qin inclue : CLDC 1.1, MIDP 2.0, WMA 1.1 MMAPI 1.1.

Elle permet aussi l'utilisation de la JSR−172 (J2ME Web Services Specification).

Lancer l'application j2me_wireless_toolkit−2_1−windows.exe. Un assistant guide l'utilisateur dans les différentes étapes de l'installation :

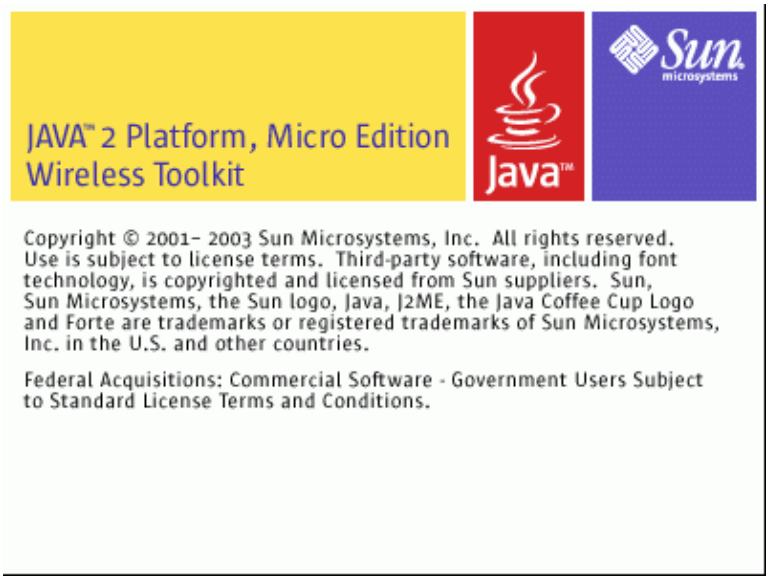
- sur la page d'accueil, cliquez sur le bouton « Next »
- sur la page « License Agreement » : lire la licence et si vous l'acceptez, cliquez sur le bouton « Yes »
- sur la page « Java Virtual Machine Location » : le programme détecte automatiquement la présence d'un JDK 1.4 ou supérieure, cliquez sur le bouton « Next »
- sur la page « Choose Destination Location » : sélectionnez le répertoire d'installation de l'application et cliquez sur le bouton « Next »
- sur la page « Select Program Folder » : saisissez ou sélectionner le dossier du menu démarrer qui va contenir les raccourcis vers l'application si celui par défaut ne convient pas. Cliquez sur le bouton « Next »
- sur la page « Start Copying files » : un résumé des options d'installation est affiché. Cliquez sur le bouton « Next »
- les fichiers de l'application sont copiés. Une fois celle ci terminée, la page « Installshield Wizard Complete » s'affiche. Cliquez sur le bouton « Finish ».

L'installation créé les répertoires suivants :

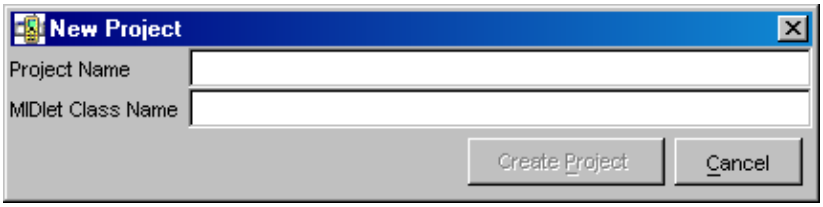
appdb\	contient les bases de données de type RMS des applications
apps\	contient les applications développées comme applications de démo
bin\	contient les outils du WTK
docs\	contient la documentation du WTK et des API
lib\	contient les bibliothèques des API

56.5.2. Premiers pas

L'outil Ktoolbar est un petit IDE qui permet de compiler, pré−vérifier, packager et exécuter des applications utilisant le profile MIDP. Il ne permet pas l'édition du code des applications : il faut utiliser un éditeur externe pour réaliser cette tâche.



La première chose à faire pour créer une application est de créer un nouveau projet. Pour cela, il faut sélectionner l'option « File/New Project » du menu ou cliquer sur le bouton « New Project » dans la barre d'outils.

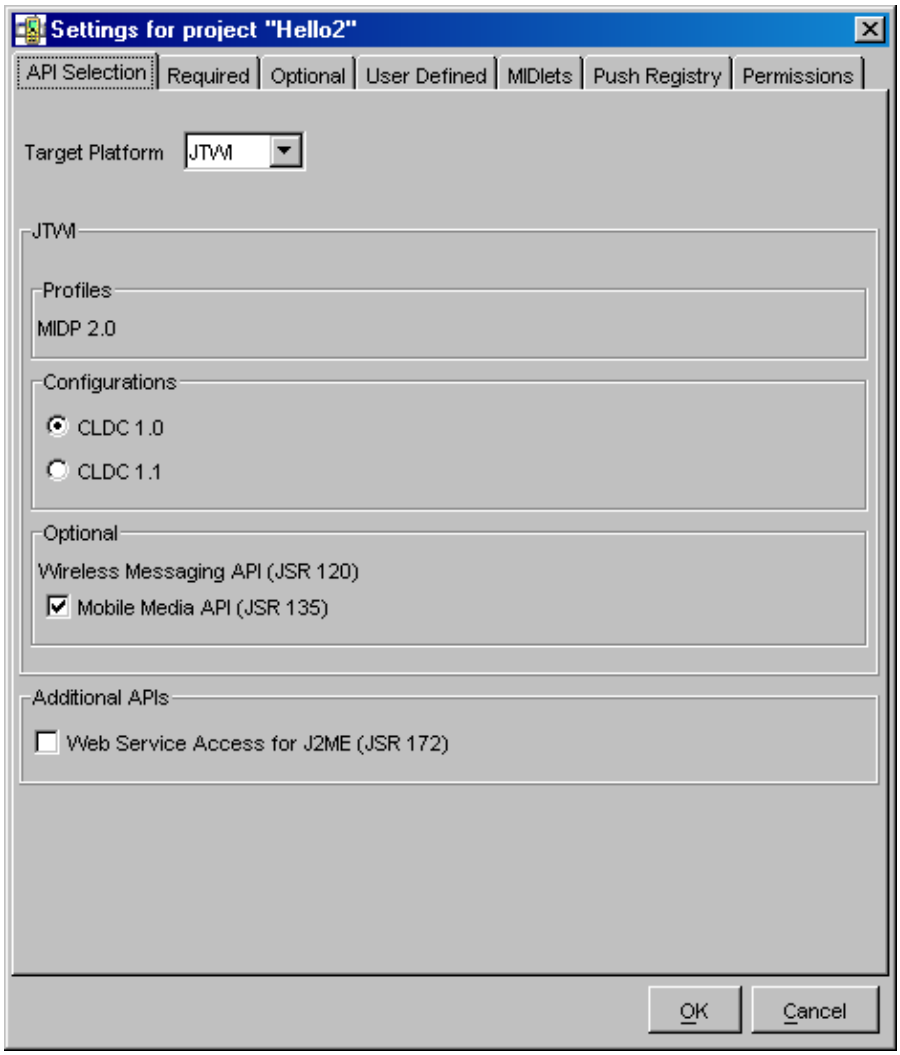


Il faut saisir le nom du projet et le nom de la classe de la Midlet.

La création du projet permet la création d'une structure de répertoires dans le sous répertoire apps du répertoire du WTK. Dans ce répertoire apps, un répertoire est créé nommé du nom du projet. Ce répertoire contient lui même plusieurs sous répertoires :

%WTK%/apps/nom_projet/bin	contient le fichier jar, jad et le fichier manifest
%WTK%/apps/nom_projet/classes	contient les classes compilées
%WTK%/apps/nom_projet/lib	contient les bibliothèques utiles à l'application
%WTK%/apps/nom_projet/res	contient les ressources utiles à l'application
%WTK%/apps/nom_projet/src	contient les sources des classes

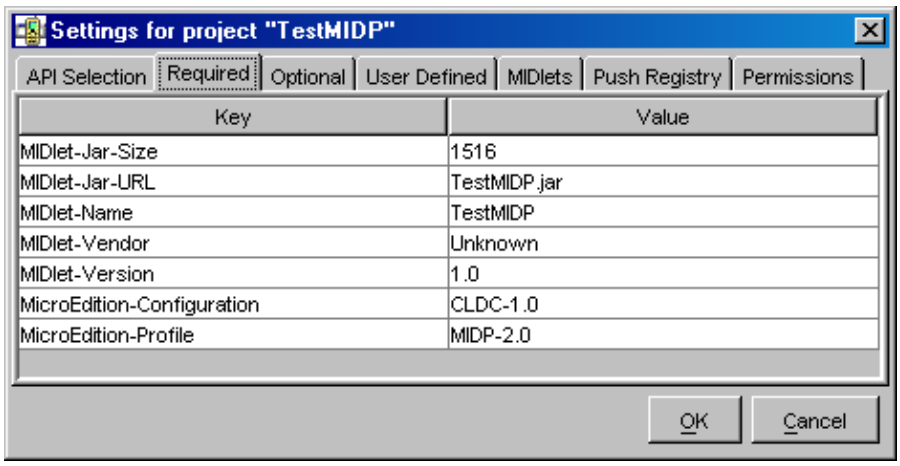
La page des propriétés du projet est différente de celle proposée dans la version 1.0. Pour l'utiliser, il faut utiliser l'option « Project/settings » ou cliquer sur le bouton « Settings » de la barre d'outils.



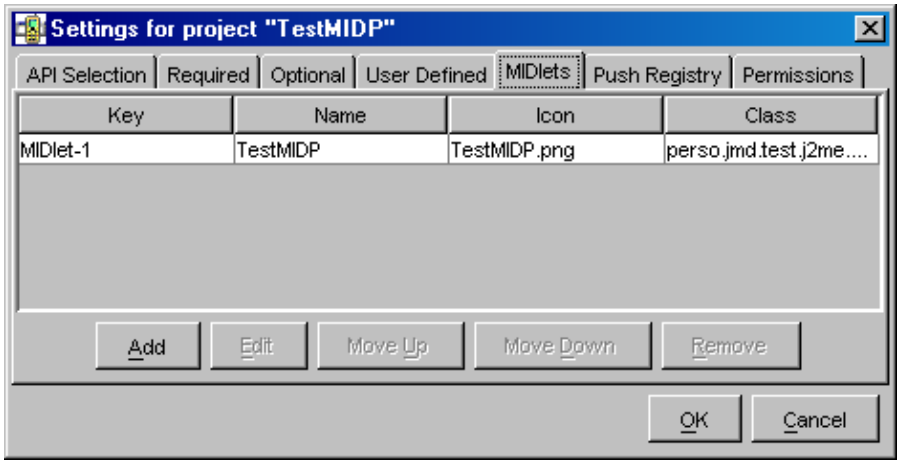
L'onglet « API sélection » permet de sélectionner la plateforme cible ainsi que les API particulières qui vont être utilisées par l'application.

Le « target plateform » permet de sélectionner le type de plate−forme cible utilisée :

- JTWI : plate−forme répondant aux spécifications de la JSR−185
- MIDP 1.0 : plate−forme composée de CLDL 1.0 et MIDP 1.0
- Custom : plate−forme personnalisée pour laquelle il faut préciser toutes les API utilisées



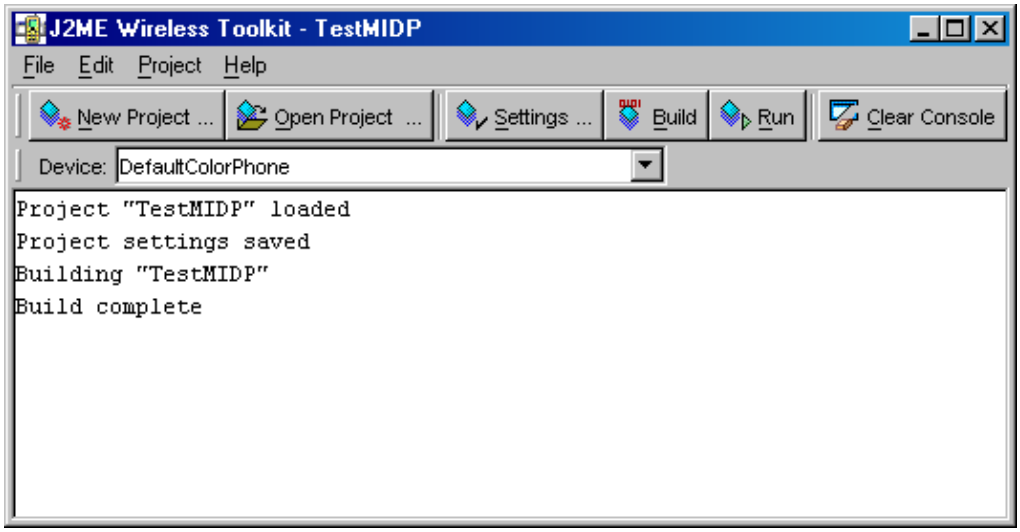
L'onglet « Required », « Optionnal » et « User defined » permet de préciser les attributs respectivement obligatoires, optionnels et particuliers à l'application dans le fichiers manifest sous la forme de paire clé/valeur.



L'onglet « Midlets » permet de saisir les Midlets qui composent la suite de Midlets de l'application.

Pour créer et éditer le code des classes qui composent l'application, il faut utiliser un outil externe dans le répertoire %WTK%/apps/nom_projet/src, en respectant la structure des répertoires correspondant aux packages des classes.

La compilation et la pré−vérification des sources se fait en utilisant l'option « Build » du menu « Project » ou en cliquant sur le bouton « Build » .

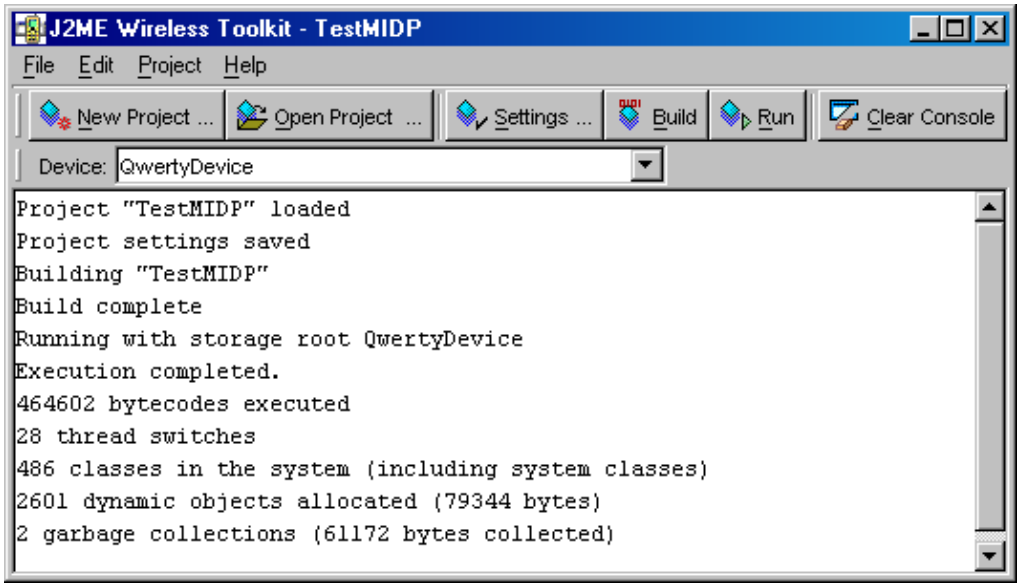


Si aucune erreur de compilation n'est détectée, il est possible d'exécuter le code en utilisant l'option « Run » du menu « Project » ou en cliquant sur le bouton « Run » de la barre d'outils.

Avant de lancer l'exécution, il est possible de sélectionner l'émulateur de périphérique (device) utilisé pour exécuter le code. Le J2ME Wireless Toolkit 2.1 est fourni avec quatre émulateurs :

- DefaultColorPhone : un téléphone mobile avec un écran couleur. C'est l'émulateur par défaut.
- DefaultGrayPhone : un téléphone mobile avec un écran monochrome
- MediaControlSkin : un téléphone mobile avec des capacités multimédia accrues (video et audio)
- QwertyDevice : un périphérique avec un clavier Qwerty

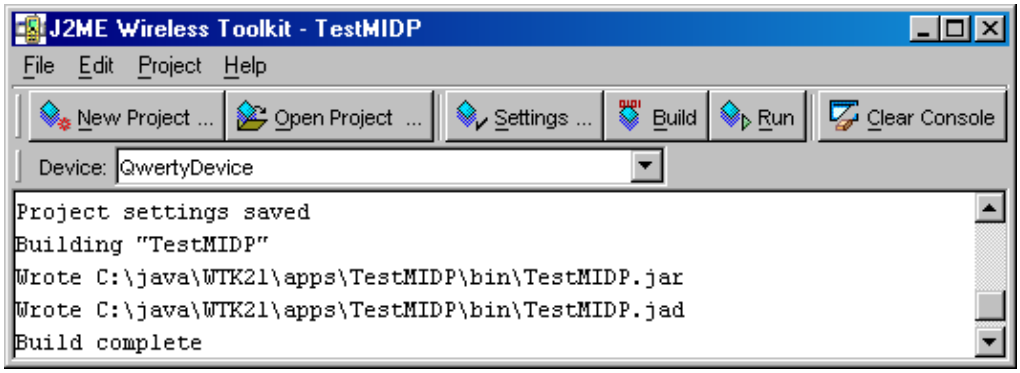
L'option « Clean » du menu « Project » permet de faire du ménage dans les fichiers temporaires générés lors des différents traitements.



Le bouton « Clear console » permet d'effacer le contenu de la console.

L'option « Package » du menu « Project » propose deux options pour packager l'application une fois celle ci mise au point :

- « Create package » : permet de créer un package sous la forme d'un fichier .jar et .jad
- « Create obfuscated package » : permet de créer un package sous la forme d'un fichier .jad et d'un fichier .jar plus compact et grâce à un outil tiers non fourni réalisant l'opération d'obscurcissement



57. CLDC

La suite de ce chapitre sera développée dans une version future de ce document

L'API du CLDC se compose de quatres packages :

- java.io : classes pour la gestion des entrées / sorties par flux
- java.lang : classes de base du langage java
- java.util : classes utilitaires notamment pour gérer les collections, la date et l'heure, ...
- javax.microedition.io : classes pour gérer des connections génériques

Ils ont des fonctionnalités semblables à ceux proposés par J2SE avec quelques restrictions, notamment il n'y a pas de gestion des nombres flottants dans CLDC 1.0.

De nombreuses classes sont définies dans J2SE et J2ME mais souvent elles possèdent moins de fonctionnalités dans l'édition mobile.

La version courant de CLDC est la 1.1 dont les spécifications sont le résultats des travaux de la JSR 139.

Ce chapitre contient plusieurs sections :

57.1. Le package java.lang

Il définit l'interface Runnable.

Il définit les classes suivantes :

Nom	Rôle
Boolean	Classe qui encapsule une valeur du type booleen
Byte	Classe qui encapsule une valeur du type byte
Character	Classe qui encapsule une valeur du type char

Class	Classe qui encapsule une classe ou une interface
Integer	Classe qui encapsule une valeur du type int
Long	Classe qui encapsule une valeur du type long
Math	Classe qui contient des méthodes statiques pour les calculs mathématiques
Object	Classe mère de toutes les classes
Runtime	Classe qui permet des intreaction avec le système d'exploitation
Short	Classe qui encapsule une valeur du type short
String	Classe qui encapsule une chaîne de caractères immuable
StringBuffer	Classe qui encapsule une chaîne de caractère
System	
Thread	Classe qui encapsule un traitement executé dans un thread
Throwable	Classe mère de toutes les exceptions et les erreurs

Il définit les exceptions suivantes : ArithmeticException, ArrayIndexOutOfBoundsException, ArrayStoreException, ClassCastException, ClassNotFoundException, Exception, IllegalAccessException, IllegalArgumentException, IllegalMonitorStateException, IllegalThreadStateException, IndexOutOfBoundsException, InstantiationException, InterruptedException, NegativeArraySizeException, NullPointerException, NumberFormatException, RuntimeException, SecurityException, StringIndexOutOfBoundsException

Il définit les erreurs suivantes : Error, OutOfMemoryError, VirtualMachineError

57.2. Le package java.io

Il définit les interfaces suivantes : DataInput, DataOutput

Il définit les classes suivantes :

Nom	Rôle
ByteArrayInputStream	Lecture d'un flux d'octets bufférisé
ByteArrayOutputStream	Ecriture d'un flux d'octets bufférisé
DataInputStream	Lecture de données stocké au format java
DataOutputStream	Ecriture de données stockées au format java
InputStream	Classe abstraite dont hérite toutes les classes gérant la lecture de flux par octets
InputStreamReader	Lecture d'octets sous la forme de caractères
OutputStream	Classe abstraite dont hérite toutes les classes gérant lécriture de flux par octets
OutputStreamWriter	Ecriture de caractères sous la forme d'octets
PrintStream	
Reader	Classe abstraite dont hérite toutes les classes gérant la lecture de flux par caractères
Writer	Classe abstraite dont hérite toutes les classes gérant l'écriture de flux par caractères

Il définit les exceptions suivantes : EOFException, InterruptedIOException, IOException, UnsupportedEncodingException, UTFDataFormatException

57.3. Le package java.util

Il définit l'interface Enumeration

Il définit les classes suivantes :

Nom	Rôle
Calendar	Classe abstraite pour manipuler les éléments d'une date
Date	Classe qui encapsule une date
Hashtable	Classe qui encapsule une collection d'éléments composés d'une par paire clé/valeur
Random	Classe qui permet de générer des nombres aléatoires
Stack	Classe qui encapsule une collection de type pile LIFO
TimeZone	Classe qui encapsule un fuseau horaire
Vector	Classe qui encapsule une collection de type tableau dynamique

Il définit les exceptions EmptyStackException et NoSuchElementException

57.4. Le package javax.microedition.io

Il définit les interface suivantes :

Nom	Rôle
Connection	Interface pour une connexion générique
ContentConnection	
Datagram	Interface pour un paquet de données
DatagramConnection	Interface pour une connexion utilisant des paquets de données
InputConnection	Interface pour une connexion entrante
OutputConnection	Interface pour une connexion sortante
StreamConnection	Interface pour un connexion utilisant un flux
StreamConnectionNotifier	

58. MIDP

C'est le premier profile qui a été développé dont l'objectif principal est le développement d'application sur des machines aux ressources et à l'interface limitées tel qu'un téléphone cellulaire. Ce profil peut aussi être utilisé pour développer des applications sur des PDA de type Palm.

L'API du MIDP se compose des API du CDLC et de trois packages :

- javax.microedition.midlet : cycle de vie de l'application
- javax.microedition.lcdui : interface homme machine
- javax.microedition.rms : persistance des données

Des informations complémentaires et le téléchargement de l'implémentation de référence de ce profil peuvent être trouvées sur le site de Sun : http://java.sun.com/products/midp/

Il existe deux versions du MIDP :

- 1.0 : la dernière révision est la 1.0.3 dont les spécifications sont issues de la JSR 37
- 2.0 : c'est la version la plus récente dont les spécifications sont issues de la JSR 118

Ce chapitre contient plusieurs sections :

58.1. Les Midlets

Les applications créées avec MIDP sont des midlets : ce sont des classes qui héritent de la classe abstraite javax.microedition.midlet.Midlet. Cette classe permet le dialogue entre le système et l'application.

Elle possède trois méthodes qui permettent de gérer le cycle de vie de l'application en fonction des trois états possibles (active, suspendue ou détruite) :

- startApp() : cette méthode est appelée à chaque démarrage ou redémarrage de l'application
- pauseApp() : cette méthode est appelée lors de la mise en pause de l'application
- destroyApp() : cette méthode est appelée lors de la destruction de l'application

Ces trois méthodes doivent obligatoirement être redéfinies.

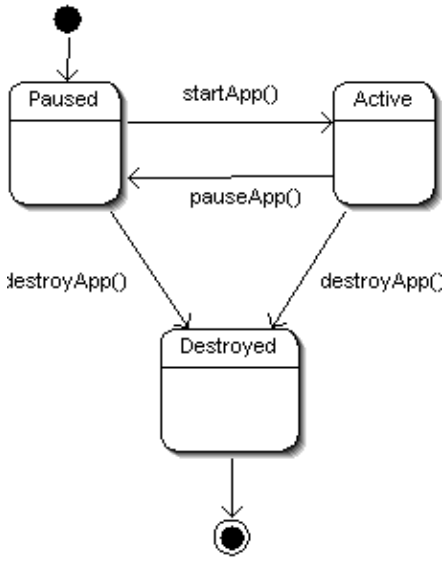
Exemple (MIDP 1.0) :

```
package perso.jmd.test.j2me;

 import javax.microedition.midlet.*;
 import javax.microedition.lcdui.*;

 public class Test extends MIDlet {

   public Test() {
   }

   public void startApp() {
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
 }
```

Le cycle de vie d'une midlet est semblable à celui d'une applet. Elle possède plusieurs états :

- paused :
- active :
- destroyed :

Le changement de l'état de la midlet peut être provoqué par l'environnement d'exécution ou la midlet.



La méthode startApp() est appelée lors du démarrage ou redémarrage de la midlet. Il est important de comprendre que cette méthode est aussi appelée lors du redémarrage de la midlet : elle peut donc être appelée plusieurs fois au cours de l'exécution de la midlet.

La méthode pauseApp() est appelée lors de mise en pause de la midlet.

La méthode destroyApp() est appelée juste avant la destruction de la midlet.

58.2. L'interface utilisateur

Les possiblités concernant l'IHM de MIDP sont très réduites pour permettre une exécution sur un maximum de machines allant du téléphone portable au PDA. Ces machines sont naturellement et physiquement pourvues de contraites forte concernant l'interface qu'ils proposent à leurs utilisateurs.

Avec le J2SE, deux API permettent le développement d'IHM : AWT et Swing. Ces deux API proposent des composants pour développer des interfaces graphiques riches de fonctionnalités avec un modèle de gestion des événements complet. Ils prennent en compte un système de pointage par souris, avec un écran couleur possèdant de nombreuses couleurs et une résolution importante.

Avec MIDP, le nombre de composants et le modèle de gestion des événements sont spartiates. Il ne prend en compte qu'un écran tactile souvent monochrome ayant une résolution très faible. Avec un clavier limité en nombres de touches et dépourvu de système de pointage, la saisie de données sur de tels appareils est particulièrement limité.

L'API pour les interfaces utilisateurs du MIDP est regroupée dans le package javax.microedition.lcdui.

Elle se compose des éléments de haut niveaux et des éléments de bas niveaux.

58.2.1. La classe Display

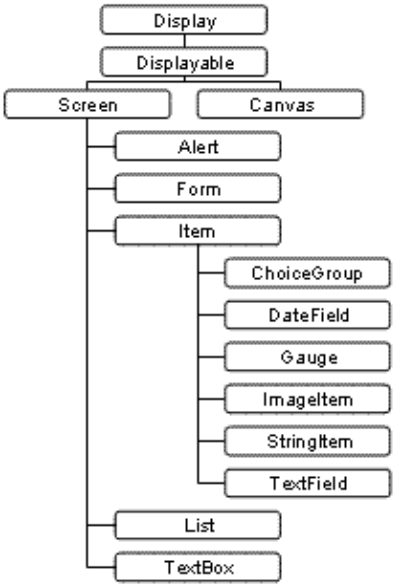
Pour pouvoir utiliser les éléments graphiques, il faut obligatoirement obtenir un objet qui encapsule l'écran. Un tel objet est du type de la classe Display. Cette classe possède des méthodes pour afficher les éléments graphiques.

La méthode statique getDisplay() renvoie une instance de la classe Display qui encapsule l'écran associé à la midlet fournie en paramètre de la méthode.

Exemple (MIDP 1.0) :

```
package perso.jmd.test.j2me;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

  private Display display;

  public Hello() {
    display = Display.getDisplay(this);
  }

  public void startApp() {
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
 }
}
```

Les éléments de l'interface graphique appartiennent à une hiérarchie d'objets : tous les éléments affichables héritent de la classe abstraite Displayable.



La classe Screen est la classe mère des éléments graphiques de haut niveau. La classe Canvas est la classe mère des éléments graphiques de bas niveau.

Il n'est pas possible d'ajouter directement un élément graphique dans un Display sans qu'il soit inclus dans un objet héritant de Displayable.

Un seul objet de type Displayable peut être affiché à la fois. La classe Display possède la méthode getCurrent() pour connaître l'objet courant affiché et la méthode setCurrent() pour afficher l'objet fourni en paramètre.

58.2.2. La classe TextBox

Ce composant permet de saisir du texte.

Exemple (MIDP 1.0) :

```
package perso.jmd.test.j2me;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

  private Display display;
  private TextBox textbox;

  public Hello() {
    display = Display.getDisplay(this);
    textbox = new TextBox("", "Bonjour", 20, 0);
  }

  public void startApp() {
    display.setCurrent(textbox);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Résultat :

sur l'émulateur Palm

sur l'émulateur de téléphone mobile

58.2.3. La classe List

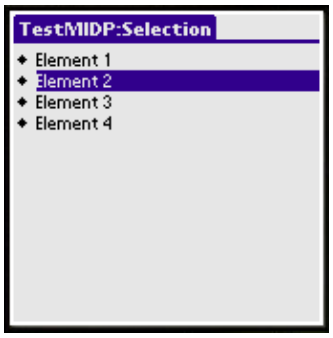
Ce composant permet la sélection d'un ou plusieurs éléments dans une liste d'éléments.

Exemple (MIDP 1.0) :

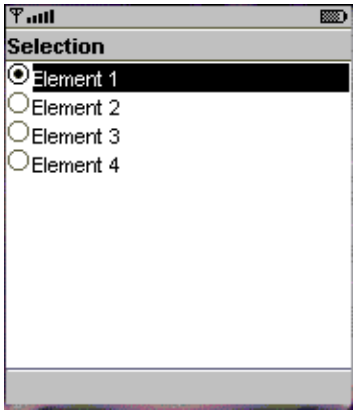
```
package perso.jmd.test.j2me;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {
  private Display display;
  private List liste;

  protected static final String[] elements = {"Element 1",
                                              "Element 2",
                                              "Element 3",
                                              "Element 4"};

  public Test() {
    display = Display.getDisplay(this);
    liste = new List("Selection", List.EXCLUSIVE, elements, null);;
  }

  public void startApp() {
    display.setCurrent(liste);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Résultat :

sur l'émulateur Palm



sur l'émulateur de téléphone mobile



La suite de cette section sera développée dans une version future de ce document

58.2.4. La classe Form

La classe From permet d'insérer dans l'élément graphique qu'elle représente d'autres éléments graphiques : cette classe sert de conteneurs. Les éléments insérés sont des objets qui héritent de la classe abstraite Item.

Exemple (MIDP 1.0) :

```
package perso.jmd.test.j2me;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

  private Display display;
  private Form mainScreen;

  public Hello() {
    display = Display.getDisplay(this);
  }

  public void startApp() {
    mainScreen = new Form("Hello");
    mainScreen.append("Bonjour");
   display.setCurrent(mainScreen);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

58.2.5. La classe Item

La classe javax.microedition.lcdui.Item est la classe mère de tous les composants graphiques qui peuvent être insérés dans un objet de type Form.

Cette classe définit seulement deux méthodes, getLabel() et setLabel() qui sont le getter et le setter pour la propriété label.

Il existe plusieurs composants qui héritent de la classe Item

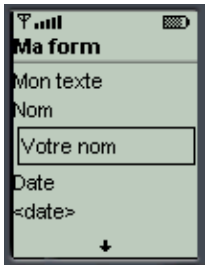
Classe	Rôle
ChoiceGroup	sélection d'un ou plusieurs éléments
DateField	affichage et saisie d'une date
Gauge	affichage d'une barre de progression
ImageItem	affichage d'une image
StringItem	affichage d'un texte
TextField	saisie d'un texte

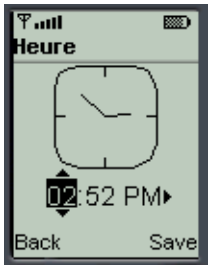
Exemple (MIDP 1.0) :

```
package perso.jmd.test.j2me;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

  private Display display;
  private Form form;
  private ChoiceGroup choiceGroup;
  private DateField dateField;
  private DateField timeField;
  private Gauge gauge;
  private StringItem stringItem;
  private TextField textField;

  public Hello() {
    display = Display.getDisplay(this);
    form = new Form("Ma form");

    String choix[] = {"Choix 1", "Choix 2"};
    stringItem = new StringItem(null,"Mon texte");
    choiceGroup = new ChoiceGroup("Sélectionner",Choice.EXCLUSIVE,choix,null);
    dateField = new DateField("Heure",DateField.TIME);
    timeField = new DateField("Date",DateField.DATE);
    gauge = new Gauge("Avancement",true,10,1);
    textField = new TextField("Nom","Votre nom",20,0);

    form.append(stringItem);
    form.append(choiceGroup);
    form.append(timeField);
    form.append(dateField);
    form.append(gauge);
    form.append(textField);
  }

  public void startApp() {
    display.setCurrent(form);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
```

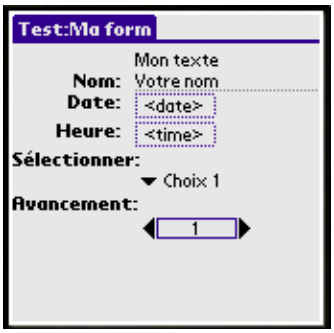
```
}
```

Résultat sur l'émulateur de téléphone mobile :





Résultat sur l'émulateur Palm OS :



58.2.6. La classe Alert

Cette classe permet d'afficher une boite de dialogue pendant un temps déterminé.

Elle possède deux constructeurs :

- un demandant le titre de l'objet
- un demandant le titre, le texte, l'image et le type de l'image

Elle possède des getters et des setters sur chacun de ces éléments.

Pour préciser le type de la boîte de dialogue, il faut utiliser une des constantes définies dans la classe AlertType dans le constructeur ou dans la méthode setType() :

Constante	type de la boite de dialogue
ALARM	informer l'utilisateur d'un événement programmé
CONFIRMATION	demander la confirmation à l'utilisateur
ERROR	informer l'utilisateur d'une erreur
INFO	informer l'utilisateur
WARNING	informer l'utilisateur d'un avertissement

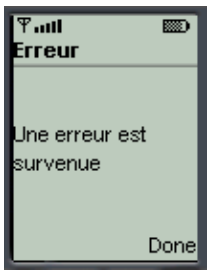
Pour afficher un objet de type Alert, il faut utiliser une version surchargée de la méthode setCurrent() de l'instance de la classe Display. Cette version nécessite deux paramètres : l'objet Alert à afficher et l'objet de type Displayable qui sera affiché lorsque l'objet Alert sera fermé.

La méthode setTimeout() qui attend un entier en paramètre permet de préciser la durée d'affichage en milliseconde de la boîte de dialogue. Pour la rendre modale, il faut lui passer le parametre Alert.FOREVER.

Exemple (MIDP 1.0) :

```
package perso.jmd.test.j2me;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {

  private Display display;
private Alert alert;
  private Form form;

  public Test() {
    display = Display.getDisplay(this);
    form = new Form("Hello");
    form.append("Bonjour");

   alert = new Alert("Erreur", "Une erreur est survenue", null, AlertType.ERROR);
   alert.setTimeout(Alert.FOREVER);
  }

  public void startApp() {
    display.setCurrent(alert, form);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Résultat sur l'émulateur de téléphone mobile:



Résultat sur l'émulateur Palm OS:

58.3. La gestion des évenements

Les interactions entre l'utilisateur et l'application se concrétisent par le traitement d'événements particuliers pour chaque action.

MIDP définit interface de type Listener pour la gestion des événements :

Interface	Rôle
CommandListener	Listener pour une activation d'une commande
ItemStateListener	Listener pour un changement d'état d'un composant(modification du texte d'une zone de texte, ...)

Cette section sera développée dans une version future de ce document

58.4. Le Stockage et la gestion des données

Avec MIDP, le méchanisme pour la persistance des données est appellé RMS (Record Management System). Il permet le stockage de données et leur accès ultérieur.

RMS propose un accès standardisé au système de stockage de la machine dans lequel s'exécute le programme. Il n'impose pas aux constructeurs la façon dont les données doivent être stockées physiquement.

Du fait de la simplicité des mécanismes utilisés, RMS ne définit qu'une seule classe : RecordStore. Cette classe ainsi que les interfaces et les exceptions qui composent RMS sont regroupées dans le package javax.microedition.rms.

Les données sont stockées dans un ensemble d'enregistrements (records). Un enregistrement est un tableau d'octets. Chaque enregistrement possède un identifiant unique nommé recordId qui permet de retrouver un enregistrement particulier.

A chaque fois qu'un ensemble de données est modifié (ajout, modification ou suppression d'un enregistrement), son numéro de version est incrémenté.

Un ensemble de données est associé à un unique ensemble composé d'une ou plusieurs Midlets (Midlet Suite).

Un ensemble de données possède un nom composé de 32 caractères maximum.

58.4.1. La classe RecordStore

L'accès aux données se fait obligatoirement en utilisant un objet de type RecordStore.

Les principales méthodes sont :

Méthode	Rôle
int addRecord(byte[],int, int)	Ajouter un nouvel enregistrement
void addRecordListener(RecordListener)	
void closeRecordStore()	Fermer l'ensemble d'enregistrement
void deleteRecord(int)	Supprimer l'enregistrement dont l'identifiant est fourni en paramètre
static void deleteRecordStore(String)	Supprimer l'ensemble d'enregistrements dont le nom est fourni en paramètre
Enumeration enumerateRecords(RecordFilter , RecordComparator, boolean)	Renvoyer une énumaration pour parcourir tout ou partie de l'ensemble

String getName()	Renvoyer le nom de l'ensemble d'enregistrements
int getNextRecordID()	Renvoyer l'identifiant du prochain enregistrement créé
int getNumRecords()	Renvoyer le nombre d'enregistrement contenu dans l'ensemble
byte[] getRecord(int)	Renvoyer l'enregistrement dont l'identifiant est fourni en paramètre
int getRecord(int, byte[], int)	Obtenir les données contenues dans un enregistrement dont l'identifiant est fourni en paramètre. Renvoie le nombre d'octets de l'enregistrement
int getRecordSize(int)	Renvoyer la taille en octets de l'enregistrement dont l'identifiant est fourni en paramètre
int getSize()	Renvoyer la taille en octets occupée par l'ensemble
static String[] listRecordStores()	Renvoyer un tableau de chaîne de caractères contenant le nom des ensembles de données associées à l'ensemble de Midlet courant
static RecordStore openRecordStore(String, boolean)	Ouvrir un ensemble de données dont le nom est fourni en paramètre. Celui ci est créé s'il n'existe pas et que le booléen est à true
void setRecord(int, byte[], int, int)	Mettre à jour l'enregistrement précisé avec les données fournies en paramètre

Pour pouvoir utiliser un ensemble d'enregistrements, il faut utiliser la méthode statique openRecordStore() en fournissant le nom de l'ensemble et un booléen qui précise si l'ensemble doit être créé au cas ou celui ci n'existe pas. Elle renvoie un objet RecordStore qui encapsule l'ensemble d'enregistrements.

L'appel de cette méthode peut lever l'exception RecordStoreNotFoundException si l'ensemble n'est pas trouvé, RecordStoreFullException si l'ensemble de données est plein ou RecordStoreException dans les autres cas problématiques.

La méthode closeRecordStore() permet de fermer un ensemble précédemment ouvert. Elle peut lever les exceptions RecordStoreNotOpenException et RecordStoreException.

La suite de cette section sera développée dans une version future de ce document

58.5. Les suites de midlets

58.6. Packager une midlet

Une application constituées d'une suite de midlets est packager sous la forme d'une archive .jar. Cette archive doit contenir un fichier manifest et tous les éléments nécessaire à l'exécution de l'application (fichiers .class et les ressources telles que les images, ...).

58.6.1. Le fichier manifest

Ce fichier contient des informations sur l'application.

Ce fichier contient une définition de propriétés utilisées par l'application. Ces propriétés sont sous la forme clé/valeur.

Plusieurs propriétés sont définis par les spécifications des midlets : celles−ci commencent par MIDLet−.

Propriétés	Rôle
MIDlet−Name	Nom de l'application
MIDlet−Version	Numéro de version de l'application
MIDlet−Vendor	Nom du fournisseur de l'application
MIDlet−Icon	Nom du fichier .png contenant l'icône de l'application
MIDlet−Description	Description de l'application
MIDlet−Info−URL	
MIDlet−Jar−URL	URL de téléchargement de fichier jar
MIDlet−Jar−Size	taille en octets du fichier .jar
MIDlet−Data−Profile	
MicroEdition−Configuration	

Il est possible de définir ces propres attributs

58.7. MIDP for Palm O.S.

MIDP for Palm O.S. est une implémentation particulière du profile MIPD pour le déploiement et l'exécution d'applications sur des machines de type Palm. Elle permet d'exécuter des applications écrites avec MIDP sur un PALM possédant une version 3.5 ou supérieure de cet O.S.

Cette implémentation remplace l'ancienne implémentation développé par Sun nommé KJava.

58.7.1. Installation

MIPD for Palm O.S. peut être téléchargé à l'URL suivante : http://java.sun.com/products/midp4palm/download.html. Il faut télécharger le fichier midp4palm−1_0.zip et le fichier midp4palm−1_0−doc.zip qui contient la documentation.

L'installation comprend une partie sur le poste de développement PC et une partie sur la machine Palm pour les tests d'exécution.

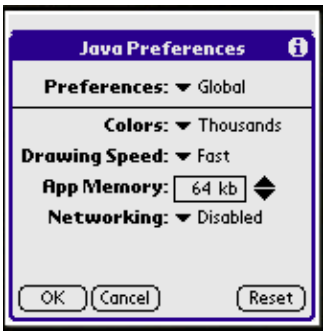
Pour pouvoir utiliser MIDP for Palm O.S., il faut déjà avoir installé CLDC et MIDP.

Il faut commencer l'installation sur le PC en dézipant les deux fichiers dans un répertoire.

Pour pouvoir exécuter les applications sur le Palm, il faut installer le fichier MIPD.prc contenu dans le répertoire PRCFiles sur le Palm en procédant comme pour toute application Palm.

En cliquant sur l'icône, on peut régler différents paramètres.

Un clic sur le bouton "Preferences" permet de modifier ces paramètres.



58.7.2. Création d'un fichier .prc

MIPD for Palm O.S. fourni un outil pour transformer les fichier .jad et .jar qui composent une application J2ME en un fichier .prc directement installable sur un Palm.

Sous Windows, il suffit d'exécuter le programme converter.bat situé dans le sous répertoire Converter du répertoire d'installation.

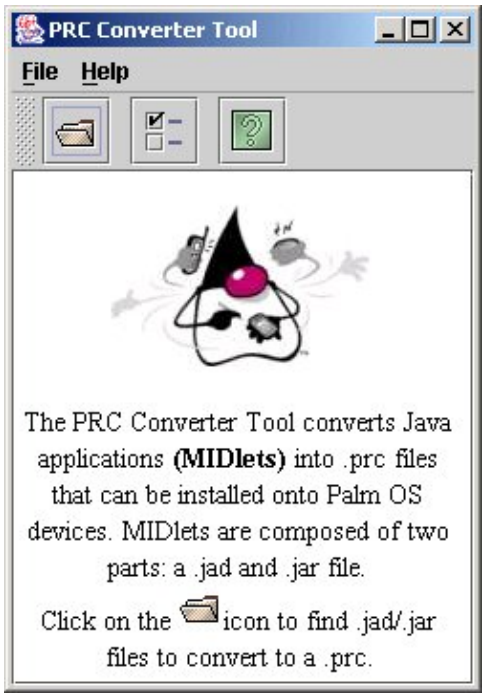
Il faut que la variable d'environnement JAVA_PATH pointe vers le répertoire d 'installation d'un JDK 1.3. minimum. Si ce n'est pas le cas, un message d'erreur est affiché.

Error: Java path is missing in your environment

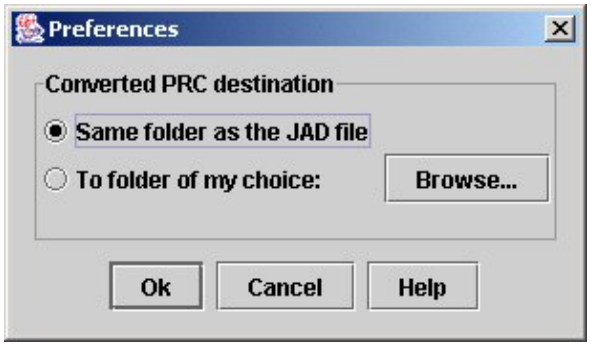
Please set JAVA_PATH to point to your Java directory

e.g. set JAVA_PATH=c:\bin\jdk1.3\

Si tout est correct, l'application se lance.

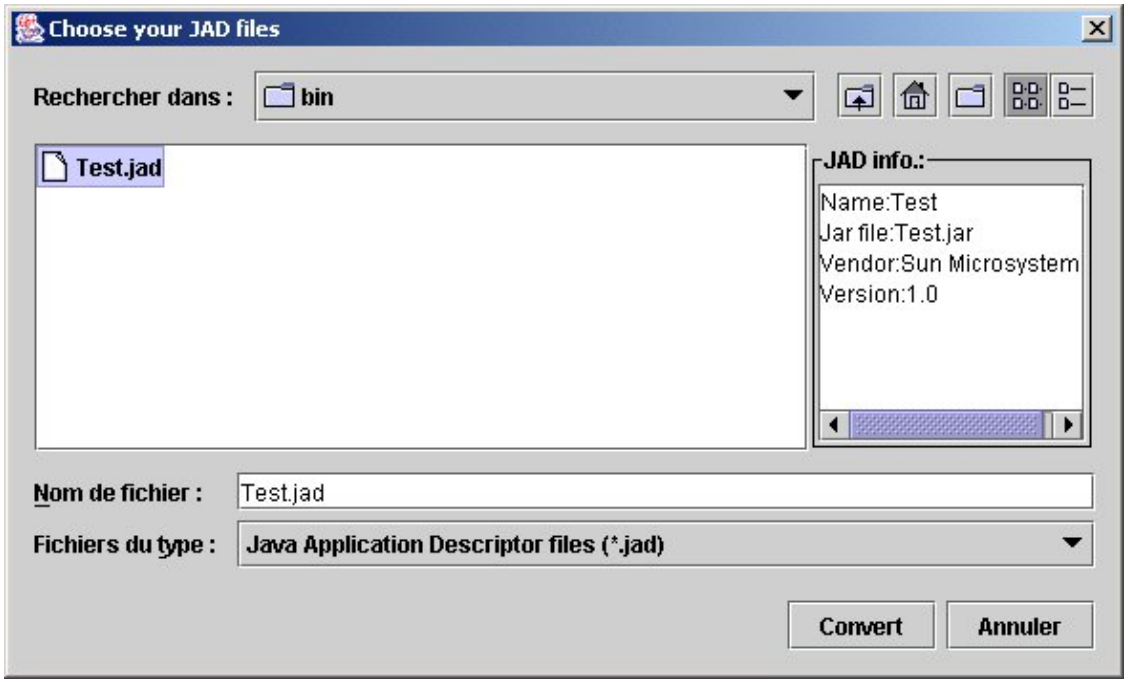


Il est possible de préciser le répertoire du ou des fichiers .prc généré en utilisant l'option "Preference" du menu "File" :



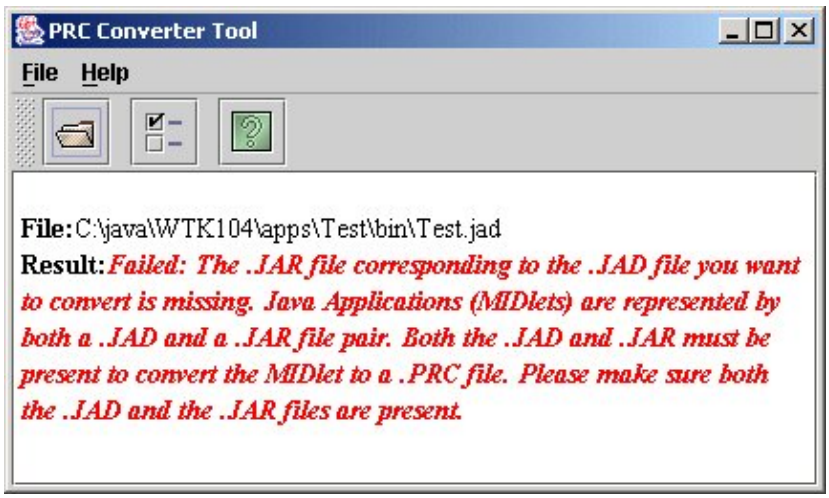
Une boîte de dialogue permet de choisir entre le même répertoire que celui qui contient le fichier .jad ou de sélectionner un répertoire quelconque.

Il suffit de cliquer sur l'icône en forme de répertoire dans la barre d'îcone pour sélectionner le fichier .jad. Les fichiers .jad et .jar de l'application doivent obligatoirement être ensemble dans le même répertoire.

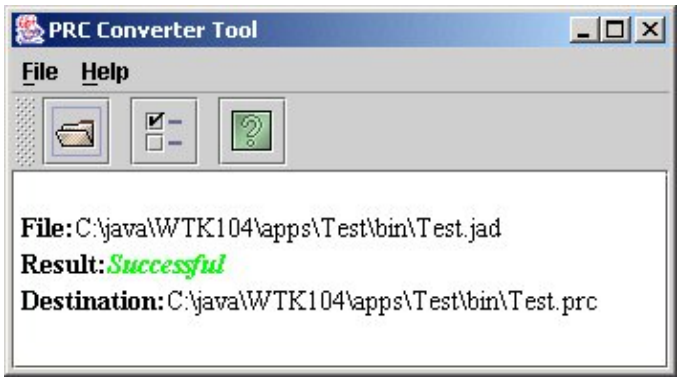


Un clic sur le bouton "Convert", lance la conversion.

Si la conversion échoue, un message d'erreur est affiché. Exemple, si le fichier .jar correspond au fichier .jad est absent, alors le message suivant est affiché :



Si toutes les opérations se sont correctement passées, alors un message récapitulatif est affiché :



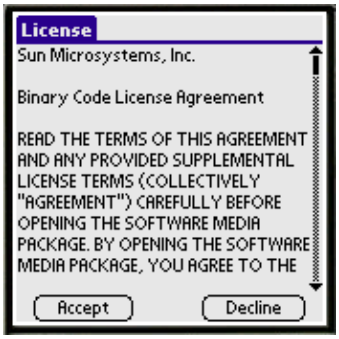
58.7.3. Installation et exécution d'une application

Une fois le fichier .prc créer, il suffit d'utiliser la procédure standard d'installation d'un tel fichier sur le Palm (ajouter le fichier dans la liste avec "l'outil d'installation" du Palm et lancer une synchronisation).

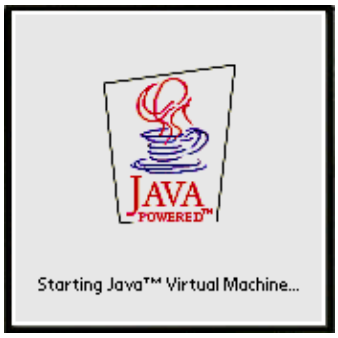
Une fois l'application installée, l'icône de l'application apparaît.

Pour exécuter l'application, il suffit comme pour une application native, de cliquer sur l'îcone.

Lors de la première exécution, il faut lire et valider la licence d'utilisation.



Une splash screen s'affiche durant le lancement de la machine virtuelle.



Puis l'application s'exécute.

59. CDC

La suite de ce chapitre sera développée dans une version future de ce document

Cette configuration se destine à l'utilisation de Java sur des machines mobiles possédant un processeur 32 bits, au moins 2Mo de RAM et une connexion au réseau.

CDC est une spécification définit par la JSR numéro 036.

La machine virtuelle utilise par le CDC est nommé CVM. Elle respecte intégralement les spécifications de la plate−forme Java 2 version 1.3.

Le CDC ne peut être utilisé seul : il faut lui adjoindre un ou plusieurs profiles qui lui sont spécifiques.

Le CDC définit aussi un ensemble d'API de base :

- java.lang
- java.util
- java.net
- java.io
- java.text
- java.security

Le contenu de ces packages est très proche de celui de la plate−forme J2SE excepté quelques exceptions et surtout la suppression de toutes les API dépréciée (deprecated).

La version 1.1 du CDC est en cours de spécification dans la JSR 218

60. Les profils du CDC

La suite de ce chapitre sera développée dans une version future de ce document

Ce chapitre contient plusieurs sections :

- Foundation profile
- Personal Basis Profile (PBP)
- Personal Profile (PP)

60.1. Foundation profile

Ce profile sert de base pour le developpement d'application sur des outils mobiles utilisant la configuration CDC tel que des Pockets PC ou des Tablets PC.

Il sert de base pour d'autres profiles.

Une partie importante de ce profil concerne les différentes formes de connection au réseau.

60.2. Personal Basis Profile (PBP)

Ce profil contient les éléments de bases pour développer une interface graphique avec le CDC et le Foundation profil.

Ce profile a été développé sous la JSR 129.

60.3. Personal Profile (PP)

Ce profile se destine au développement d'applications sur des PDA disposant de ressources importantes tel que les Pockets PC. Ce profil permet notamment le développement d'IHM évoluée.

Le Personal Basis Profile est un sous ensemble du Personal Profile.

http://java.sun.com/products/personalprofile/

Ce profile a été développé sous la JSR 62.

61. Les autres technologies pour les applications mobiles

La suite de ce chapitre sera développée dans une version future de ce document

Ce chapitre contient plusieurs sections :

- KJava
- PDAP (PDA Profile)
- PersonalJava
- Java Phone
- JavaCard
- Embedded Java
- Waba, Super Waba, Visual Waba

61.1. KJava

Ce n'est pas un profil officiel mais un développement proposé par Sun pour réaliser des développements sur des machines de type Palm.

KJava n'est plus supporté par Sun. Il faut utiliser MIDP for Palm O.S. à la place.

61.2. PDAP (PDA Profile)

Ce profile permet le développement d'application sur PDA en tenant compte notamment de l'accès aux données. Il utilise la configuration CLDC.

Il propose deux packages optionels :

- Personal Information Management (PIM) : pour standardiser l'accès aux données personnelles stockées dans la plupart des PDA tels que le carnet d'adresse, l'agenda, le boc−notes, ...
- File Connection (FC) : pour permettre l'accès aux données stockées dans un système de fichiers externe tel que les cartes mémoires

Les spécifications de ce profile sont en cours de développement sous la JSR 075 : http://jcp.org/en/jsr/detail?id=075 (PDA Optional Packages for the J2ME Platform)

61.3. PersonalJava

http://java.sun.com/products/personaljava/

La dernière version de ces spécifications est la 1.2.

PersonalJava est composé de packages obligatoires et facultatifs.

Sun propose un outil pour émuler un environnement d'exécution pour des applications développer avec PersonalJava : PJEE (PersonalJava Emulation Environment).

Ce profile a été abandonné au profit d'un ensemble de profils qui respecte mieux le découpage des rôles de J2ME : CDC, Foundation profile et Personal Profile.

61.4. Java Phone

http://java.sun.com/products/javaphone/

61.5. JavaCard

http://java.sun.com/products/javacard/

61.6. Embedded Java

Cette technologie n'est plus supportée par Sun qui propose en remplacement CLDC et MIDP de la plate−forme J2ME.

http://java.sun.com/products/embeddedjava/

61.7. Waba, Super Waba, Visual Waba

Partie 7 : Annexes

Annexe A : GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND définiTIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or Back−Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text

editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly−accessible computer−network location containing a complete Transparent copy of the Document, free of added material, which the general network−using public has access to download anonymously at no charge using public−standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative définition of a standard.

You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self−contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version

ever published (not as a draft) by the Free Software Foundation.

Annexe B : Glossaire

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X
Y	Z				

A

API (Application Programming Interface)	Une API est une bibliothèque qui regroupe des fonctions sous forme de classes pouvant être utilisées pour développer.
Applet	C'est une petite application java compilée, incluse dans une page html, qui est chargée par un navigateur et qui est exécutée sous le contrôle de celui ci. Pour des raisons de sécurité, par défaut, les applets ont des possibilités très resteintes.
AWT (Abstract Window Toolkit)	C'est une bibliothèque qui regroupe des classes pour développer des interfaces graphiques. Ces composants sont dit "lourds" car ils utilisent les composants du système sur lequel ils s'executent. Ainsi, le nombre des composants est volontairement resteints pour ne conserver que les composants présents sur tous les systèmes.

B

BDK (Beans Development Kit)	C'est un outil fourni par Sun qui permet d'assembler des beans de façon graphique pour générer des applications.
Bean	C'est un composant réutilisable. Il possède souvent une interface graphique mais pas obligatoirement.
BluePrints	Ce sont des documents proposés par Sun pour faciliter le développement avec Java (exemple de code, conseils, design patterns, FAQ, ...)
BMP (Bean Managed Persistence)	Type d''EJB entité dont la persistance est à la charge du code qu'il contient
Byte code	Un programme source java est compilé en byte code. C'est un langage machine indépendant du processeur. Le byte code est ensuite traduit par la machine virtuelle en langage machine compréhensible par le système ou il s'execute. Ceci permet de rendre java indépendant de tout système.

C

CLASSPATH	

	C'est une variable d'environement qui contient les répertoires contenant des bibliothèques utilisables pour la compilation et l'execution du code.
CLDC (Connected Limited Device Configuration)	Configuration J2ME pour des appareils possédant de faibles ressources et une interface utilisateur réduite tels que des téléphones mobiles, des pagers, des PDA, etc ...
CDC (Connected Device Configuration)	Configuration J2ME pour des appareils embarqués possédant certaines ressources et une connexion à internet tels que des set top box, certains PDA haut de gamme, des systèmes de navigations pour voiture, etc ...
CMP (Container Managed Persistence)	Type d'EJB entité dont la persistance est assurée par le conteneur
CORBA (Common Object Request Broker Architecture)	C'est un modèle d'objets distribués indépendant du langage de développement des objets dont les spécifications sont fournies par l'OMG.
Core class	C'est une classe standard qui est disponible sur tous les systèmes ou tourne Java.
Core packages	C'est l'ensemble des packages qui composent les API de la plate−forme Java.

D

Deprecated	Terme anglais qui peut être attribué une classe, une interface, un constructeur, une méthode ou un attribut lorque celle ci ne doivent plus être utilisés car Sun ne garantit pas que cet élément sera encore présent dans les prochaine versions de l'API.
DOM (Document Object Model)	Spécification et API pour représenter et parcourir un document XML sous la forme d'un arbre en mémoire

E

EAR (Enterprise ARchive)	Archive qui contient une application J2EE
EJB (Entreprise Java Bean)	Les EJB sont des composants métier qui répondent à des spécifications précises. Il existe deux types d'EJB : EJB Entity qui s'occupe de la persistence des données et EJB session qui gère les traitements. Les EJB doivent s'executer sur un serveur dans un conteneur d'EJB.
Exception	C'est un mécanisme qui permet de gérer les anomalies et les erreurs détectées dans une application en facilitant leur détections et leur traitement. Les exceptions sont largement utilisées et intégrées dans le langage java pour accroitre la sécurité du code.

F

G

Garbage Collector (Ramasse miettes)	C'est un mécanisme intégré à la machine virtuelle qui récupère automatiquement la mémoire inutilisée en restituant les zones de mémoire laissées libres suite à la destruction des objets.

H

HotJava	Navigateur web de Sun écrit en Java
HTML (HyperText Markup Language)	Langage à base de balises pour formatter une page web affichée dans un navigateur

I

IDL (Interface définition Language)	Langage qui permet de définir des objets devant être utilisé avec CORBA
IIOP (Internet Iter Orb Protocole)	Protocole pour faire communiquer des objets CORBA
Interface	C'est une définition de méthodes et de variables de classes que doivent respecter les classes qui l'implémente. Une classe peut implémenter plusieurs interfaces. La classe doit définir toutes les méthodes des interfaces sinon elle est abstraite.
Introspection	Fonction qui permet d'obtenir dynamiquement les entités (champs et méthodes) qui composent un objet

J

J2EE (Java 2 Entreprise Edition)	C'est une version du JDK qui contient la version standard plus un ensemble de plusieurs API permettant le développement d'applications destinées aux entreprises : EJB, Servlet, JSP, JNDI, JMS, JTA, JTS, ...
J2ME (Java 2 Micro Edition)	C'est une version du JDK qui contient le nécessaire pour développer des applications capables de fonctionner dans des environnements limités tels que les assistants personnels (PDA), les téléphones portables ou les systèmes de navigation embarqués
J2SE (Java 2 Standard Edition)	C'est une version du JDK qui contient le nécessaire pour développer des applications et des applets.
JAAS (Java Authentication and Authorization Service)	API qui permet d'authentifier un utilisateur et de leur accorder des droits d'accès
JAI (Java Advanced Imaging)	API dédié à l'utilisation et à la transformation d'images
JAR (Java ARchive)	Technique qui permet d'archiver avec ou sans compression des classes java et des ressources dans un fichier unique de façon indépendante de toute plate−forme. Ce format supporte aussi la signature électronique.
Java Media API	Regroupement d'API pour le multimédia
Java One	Conférence des développeurs Java périodiquement organisée par Sun
JavaHelp	Système d'aide pour les utilisateurs d'application entièrement écrit en java
JavaMail	API pour utiliser la messagerie électronique (e mail)
Java Web Start	Outil qui permet d'utiliser une application client par téléchargement automatique via le réseau
Java XML Pack	Regroupe des API pour l'utilisation de XML avec Java

JAXB (Java API for XML Binding)	API pour faciliter la persistence entre objets java et document XML
JAXM (Java API for XML Messaging)	API pour échanger des messages XML notamment avec les services web
JAXP (Java API for XML Processing)	API pour parcourir un document XML (DOM et SAX) et le transformer avec XSLT
JAXR (Java API for XML Registries)	API pour utiliser les services d'annuaires pour les services web (UDDI)
JAX−RPC (Java API for XML Remote Procedure Calls)	API pour utiliser l'appelle de méthodes distantes via SOAP
JCA (Java Connector Architecture)	Spécification pour normaliser le développement de connecteurs vers des progiciels
JCP (Java Community Process)	Processus utilisé par Sun et de nombreux partenaires pour gèrer les évolutions de java et de ces API
JDBC (Java Data Base Connectivity)	C'est une API qui permet un accès à des bases de données tout en restant indépendante de celles ci. Un driver spécifique à la base utilisée permet d'assurer cette indépendance car le code Java reste le même.
JDC (Java Delevopper Connection)	C'est un service en ligne proposé gratuitement par Sun après enregistrement qui propose de nombreuses ressources sur java (tutorial, cours, information, mailing ...).
JDO (Java Data Objects)	API et spécification pour faciliter le mapping entre objet java et une source de données
JDK (Java Development Kit)	C'est l'environnement de développement Java.Il existe plusieurs versions majeures : 1.0, 1.1, 1.2 (aussi appelée Java 2) et 1.3. Tous les outils fournis sont à utiliser avec une ligne de commandes.
JFC (Java Foundation Class)	C'est un ensemble de classes qui permet de développer des interfaces graphiques plus riches et plus complets qu'avec AWT
JIT Compiler (Just In Time Compiler)	C'est un compiltateur qui compile le byte−code à la volée lors de l'exécution des programme pour améliorer les performances.
JMS (Java Messaging Service)	C'est une API qui permet l'échange de messages assynchrones entre applications en utilisant un MOM (Middleware Oriented Message)
JMX (Java Management eXtension)	API et spécification qui permet de développer un système d'administration d'application à distance via le réseau
JNDI (Java Naming and Directory Interface)	C'est une bibliothèque qui permet un accès aux annuaires de l'entreprise. Plusieurs protocoles sont supportés : LDAP, DNS, NIS et NDS.
JNI (Java Native Interface)	C'est un API qui normalise et permet les appels de code natif dans une application java.
JRE (Java Runtime Environment)	C'est l'environnement d'execution des programmes Java.
JSDK (Java Servlet Development Kit)	C'est un ensemble de deux packages qui permettent le développement des servlets.
JSP (Java Server Page)	C'est une technologie comparable aux ASP de Microsoft mais utilisant Java. C'est une page HTML enrichie de tag JSP et de code Java. Une JSP est traduite en servlet pour être executée. Ceci permet de séparer la logique de présentation et la logique de traitement contenu dans un composant serveur tel que des servlets, des EJB ou des beans.
JSR (Java Specification Request)	Démande d'évolution ou d'ajout des API Java traité par le JCP
JSSE (Java Secure Socket Extension)	API permettant l'utilisation du protocole SSL pour des échange HTTP sécurisé

JSTL (Java Standard Tag Library)	Bibliothèque de tags JSP standard
JTS (Java Transaction Service)	API pour utiliser les transactions
JTWI	Spécification issue de la JSR 185 visant à définir un environnement d'exécution utilisant CLDC, MIDP et plusieurs profiles de façon homogène
JUG (Java User Group)	Groupe d'utilisateurs Java
JVM (Java Virtual Machine)	C'est la machine virtuelle dans laquelle s'éxecute le code Java. C'est une application native dépendante du système d'exploitation sur laquelle elle s'execute. Elle répond à des normes dictées par Sun pour assurer la portabilité du langage. Il en existe plusieurs développées par plusieurs éditeurs notamment Sun, IBM, Borland, Microsoft, ...

K

L

Layout Manager (gestionnaire de présentation)	Les layout manager sont des classes qui gèrent la disposition des composants d'une interface graphique sans utiliser des coordonnées.
LDAP	Protocole qui permet d'accéder à un annuaire d'entreprise

M

Message Driven Bean	Type d'EJB qui traite les messages reçus d'un MOM de façon assynchrone
Midlet	Application mobile développée avec CLDC et MIDP
MIDP (Mobile Information Device Profile)	Profil utilisé avec la configuration CLDC pour le développement d'applications mobiles sous la forme de Midlets
MOM (Middleware Oriented Message)	Outil qui permet l'échange de messages entre applications
MVC (Model View Controler)	Modèle de conception largement répandu qui permet de séparer l'interface graphique, les traitements et les données manipulées

N

O

ODBC (Open Database Connectivity)	API et spécifications de Microsoft pour l'accès aux bases de données sous Windows
ORB	Middleware orienté objet pour mettre en oeuvre CORBA

P

Package (Paquetage)	Il permettent des regrouper des classes par critères. Ils impliquent une structuration des classes dans une arborescence correspondant au nom

	donné au package.

Q

R

Ramasse miette	
RI (Reference Implementation)	Implementation de référence proposée par Sun d'une spécification particulière
RMI (Remote Method Invocation)	C'est une technologie développée par Sun qui permet de faire des appels d'objets distants. Cette technologie est plus facile à mettre en oeuvre que Corba mais elle ne peut appeler que des objets java.

S

Sandbox (bac à sable)	Il désigne un ensemble des fonctionnalités et d'objets qui assure la sécurité des applications. Son composant principal est le gestionnaire de sécurité. Par exemple, Il empeche par défaut à une applet d'accéder aux ressources du système.
SAX (Simple API for XML)	API pour traiter séquentiellement un document XML en utilisant des événements
Serialization	Fonction qui permet à un objet d'envoyer son état dans un flux pour permettre sa persistance ou son envoi à travers un reseau par exemple
Servlet	C'est un composant Java qui s'exécute côté serveur dans un environnement dédié pour répondre à des requêtes. L'usage le plus fréquent est la génération dynamique de page Web. On les compare souvent aux applets qui s'éxecutent côté client mais elles n'ont pas d'interface graphique.
SOAP (Simple Object Access Protocol)	Protocole des services web pour l'échange de messages et l'appel de méthodes distances grace à XML
SQL/J	spécification qui permet d'imbriquer du code SQL dans du code Java
Swing	Framework pour le développement d'interfaces graphiques composé de composants légers

T

Taglibs	Bibliothèques de tags personnalisés utilisés dans les JSP

U

V

W

WAR (Web ARchive)	Archive qui contient une application web
webapp (web application)	Application web reposant sur les servlets et les JSP

X

Y

Z