Programmation Orienté Objet
- JAVA -

Chapitre 2
Environnements de développement JAVA

bekkalimohammed@gmail.com

Plan

I. Introduction
II. Les caractéristiques de Java
III. Les modes de programmation
IV. JDK / JRE
V. Les éditions de Java
VI. Les API et Outils de développement
VII. Les conventions d’écriture
VIII. Modificateurs et visibilité

Introduction

Java est un langage de programmation orienté objet développé par Sun Microsystems. Sa première version a été publié en 1995.

Les caractéristiques

- Java est interprété (WORA)
- Java est portable : il est indépendant de toute plate-forme
- Java est orienté objet
- Java est simple
- Java est fortement typé
- Java assure la gestion de la mémoire
- Java est sure
- Java est économe
- Java est multitâche

Les caractéristiques

Java est interprété:

Le code 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).

Les caractéristiques

Java est portable:

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.

Les caractéristiques

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,...).

Les caractéristiques

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, ...

Les caractéristiques

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.

Les caractéristiques

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.

Les caractéristiques

Java est sure:

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 caractéristiques

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 modes de programmation

Il existe 2 types de programmes avec la version standard de Java : les applets et les applications.

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 plugin sachant gérer les applets Java, ou testées avec l'**AppletViewer**.

Le JDK (Java Developpement Kit)

Définition:

Le JDK est l'ensemble des programmes nécessaires pour le développement des applications Java. Il regroupe ainsi les programme javac.exe, java.exe et appletviewer.exe pour exécuter les applets, ainsi que d'autres classes et utilitaires de développements.

Le JDK (Java Developpement Kit)

Evolution:

Depuis 1995 jusqu'à aujourd'hui le JDK n'a cessé d'évoluer et d'être étendu de version en version. En se retrouve finalement avec 7 versions.

- Le JDK 1.0 (lancé en 1995 et constitué de quelques 201 classes et interfaces) a subi quelques modifications jusqu'à sa dernière version 1.0.2.
- Le JDK 1.1 (503 classes et interfaces)
- Le JDK 1.2 (1520 classes et interfaces) a subi lui-même plusieurs modifications jusqu'à sa dernière version 1.1.8.
- Le JDK 1.3 (1840 classes et interfaces) annoncé en décembre 1998 a évolué avec sa dernière version 1.2.2.

Le JDK (Java Developpement Kit)

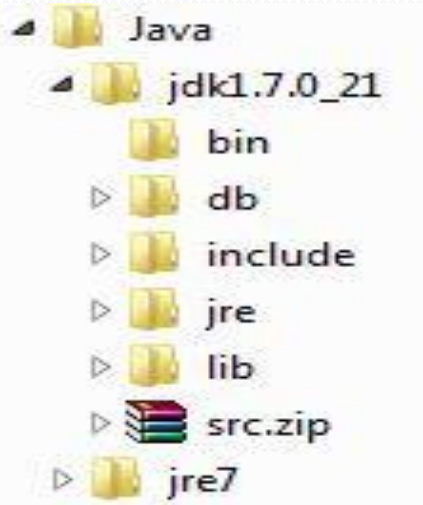
Evolution:

- Le JDK 1.4 (Nom de code **Merlin**, avec 135 packages et 2990 classes et interfaces).
- Le JDK 1.5 (ou JDK 5, Nom de code **Tigre**, avec 166 packages et 3280 classes et interfaces).
- Le JDK 1.6 (Nom de code **Mustang**, avec 202 packages et 3780 classes et interfaces).
- Le JDK 1.7 dernière version jusqu'à maintenant.

Le JDK (Java Developpement Kit)

Installation sur le disque:

- Généralement le JDK est délivré ou téléchargé sous forme d'un programme dont l'exécution conduit l'installation du kit.
- Si l'on prend par exemple du JDK 1.7.0.17, son installation par défaut créera l'arborescence suivante (de quelques 227 Mo)



Tout les programmes qu'on a besoin pour développer (java, javac, ..) se trouveront alors dans le répertoire : jdk1.7.0.17\bin qu'il est possible d'ajouter au path pour être accessible depuis n'importe quel répertoire du disque.

Le JRE (Java Runtime Environnement)

Définition:

Le JRE (Java Runtime Environnement) 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.

Les éditions de Java

Sun définit trois plateformes d'exécution (ou éditions) pour Java pour des cibles distinctes selon les besoins des applications à développer :

- Java Standard Edition (J2SE / Java SE) : environnement d'exécution et ensemble complet d'API pour des applications de type desktop. Cette plate forme sert de base en tout ou partie aux autres plateformes.
- Java Enterprise Edition (J2EE / Java EE) : environnement d'exécution reposant intégralement sur Java SE pour le développement d'applications d'entreprises.
- Java Micro Edition (J2ME / Java ME) : environnement d'exécution et API pour le développement d'applications sur appareils mobiles et embarqués dont les capacités ne permettent pas la mise en œuvre de Java SE.

Les éditions de Java

La séparation en trois plateformes permet au développeur de mieux cibler l'environnement d'exécution et de faire évoluer les plateformes de façon plus indépendante.

Avec différentes éditions, les types d'applications qui peuvent être développées en Java sont nombreux et variés :

- Applications desktop
- Applications web : servlets/JSP, portlets, applets
- Applications pour appareil mobile (MIDP) : midlets
- Applications pour appareil embarqué (CDC) : Xlets
- Applications pour carte à puce (Javacard) : applets Javacard

Les API et outils

Ainsi l'ensemble des API et des outils utilisables est énorme et évolue très rapidement.

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

Les outils libres (les plus connus)

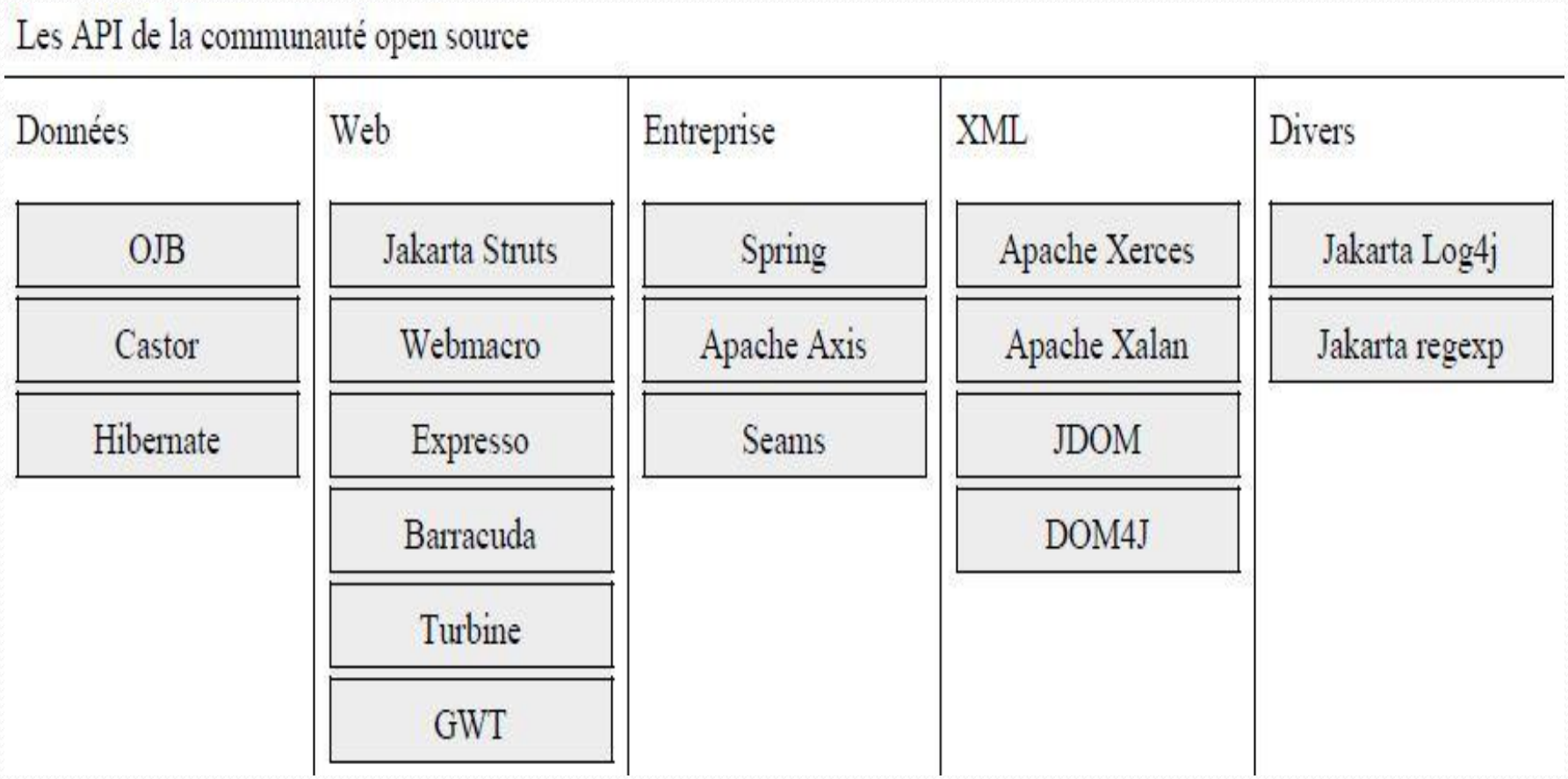
Jakarta Tomcat	Jakarta Ant	JBoss	Apache Axis
JUnit	Eclipse	NetBeans	Maven

Les API et outils

Les autres API

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

Les API et outils



Les différences entre Java et JavaScript

	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	Exécuté dans la JVM du navigateur	Exécuté 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

Les packages

Définition:

Le JDK dispose de plusieurs classes prête a être utilisés. Ces classes sont délivrées avec le JDK ou le JRE et elles sont organisées en des arborescences de répertoire. Ainsi par exemple la classe **String** se trouve dans le sous répertoire java\lang . Ce dernier est appelé un paquetage ou **package**.

Exemple de package:

java.util, java.awt, javax.swing …

Les packages

Remarque:

Dans un programme java, pour utiliser une classe d'un certain package par exemple la classe **Vector** du package java.util, on démarre le programme avec la ligne:

import java.util.*;

Ce qui signifie que toutes les classes du package sont accessible ou encore:

import java.util.vector;

Pour n'accéder que à la classe Vector

Les conventions d'écriture

1- Les noms de packages sont entièrement en minuscule. Exemples :

java.awt

javax.swing

javax.swing.filechooser

2- Un nom de classe est une séquence de mots dont le premier caractère de chaque mot est en majuscule et les autres en minuscule. Exemples :

String

StringBuffer

ComboBoxEditor

Les conventions d'écriture

3- Un nom de méthode est une séquence de mots dont le premier caractère de chaque mot est en majuscule et les autres en minuscule sauf le premier mot qui est entièrement en minuscule. Exemples :

append

toString

deleteCharAt

4- Une propriété est en principe un membre privée donc non accessible directement et par suite on peut lui choisir un nom librement. Sinon on lui applique la même règle que celle d'une méthode.

Les conventions d'écriture

5- Une constante (**final**) est une séquence de mots majuscules séparés par un blanc souligné « _ »

PI

MIN_VALUE

MAX_VALUE

6- Les primitives (types de base ou type primitif) et les mots clés sont en minuscule :

byte, int, …

while, for, if

this, super, try, cach, length, …

class, extends, implements, null, …

Structure générale d'un programme Java

Le squelette générale d'un programme Java se présente comme suit (chaque classe dans fichier séparé et portant le même nom que celui de la classe)

```
class NomDeClasse1 {
      // Variables d'instance de la classe
      // Méthodes de la classe
}

class NomDeClasse2 {
      // Variables d'instance de la classe
      // Méthodes de la classe
}
...
class NomDeClassePrincipale {
      // Variables d'instance de la classe
      // Méthodes de la classe
      public static void main(String args[])
      {
            // Code du programme principal
      }
}
```

Modificateurs et visibilité

Il existe 5 types de modificateurs qui peuvent être associés à une données ou une méthode: modificateurs de synchronisation, de visibilité, de permanence, de constance et d'abstraction.

<table>
<tr><th>1- Synchronisation</th><th>2- Visibilité</th><th>3- Permanence</th><th>4- Constance</th><th>Type</th><th>Nom</th></tr>
<tr><td>synchronized
(seulement avec les méthodes)</td><td>public
private
protected</td><td>static</td><td>final</td><td rowspan="3">void
int
…</td><td rowspan="3"></td></tr>
<tr><td colspan="4">5- Abstraction</td></tr>
<tr><td colspan="4">Abstract
(seulement les méthodes qui ne sont pas synchronized, static, final ou private)</td></tr>
</table>

Modificateurs et visibilité

Modificateur synchronized:

Permet de mettre en place une méthode ou un bloc de programme verrouillé par l'intermédiaire de mécanisme de moniteur. Une méthode est **synchronized** est une méthode dont l 'accès sur un même objet est réalisé en exclusion mutuelle.

Modificateurs et visibilité

Modificateur private, protected et public:

- Une donnée ou méthode **private** est inaccessible depuis l'extérieur de la classe où elle est définie même dans une classe dérivée.
- Une donnée ou méthode **public** est accessible depuis l'extérieur de la classe où elle est définie.
- Une donnée ou méthode **protected** est protégé contre tous accès externe comme private sauf à partir des classes dérivées. Une donnée ou méthode protected est donc accessible dans une classe fille.

Modificateurs et visibilité

Remarque:

Lorsqu'on redéfinie une méthode dans une classe fille, il est obligatoire de conserver sont modificateur de visibilité ou d'utiliser un privilège d'accès plus fort, Càd:

- Une méthode **public** reste **public**
- Une méthode **protected** reste **protected** ou devient **public**
- Une méthode **private** reste **private** ou devient **protected** ou **public**

Modificateurs et visibilité

Modificateur static:

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**.

Les données static sont appelées **variables de classe** et les méthodes static sont également appelées **méthodes de classe**.

Modificateurs et visibilité

Remarque:

Puisque les données non-static d'une classe n'ont d'existence que lorsqu'on a instancier un objet de la classe et inversement une méthode static existe sans avoir besoin de créer d'instance, **une méthode static ne peut pas donc accéder à une donnée non-static ou à une autre méthode non-static**.

Modificateurs et visibilité

Exemple :

```
class Class1 {
        static int x = 20;
        int y = 30;
        static void p1() {
                x = x + 1 ; // correcte
                y = y * 2 ; // incorrecte
                p2() ; // incorrecte
        }
        void p2() { // méthode non-static
                x ++ ; // correcte
                y ++ ; // correcte
                p1() ; // correcte
        }
}
```

Modificateurs et visibilité

Modificateur final:

Les données **final** sont des constantes. Ces données sont généralement définies en plus public et static afin d'être accessible depuis l'extérieur de la classe et directement par l'intermédiaire du nom de celle-ci sans avoir besoin de créer une instance de la classe.

Exemple :

```
class Constantes {
        public static final int CONST1 = 20;
        public static final double PI = 3.14;
}
```

Modificateurs et visibilité

Remarque:

- Une méthode déclarée final ne peut pas être redéfinie dans une sous classe.
- Lorsque le modificateur final est ajouté à une classe, il est interdit de créer une classe qui en hérite.

Modificateurs et visibilité

Modificateur abstract:

Ce modificateur permet de définir une méthode abstraite. Càd une méthode dont le corps n'est pas défini. Une classe qui comporte une méthode abstraite doit, elle aussi, être définie abstraite. Il s'agit d'une classe de spécification qui ne peut être directement instancié. Elle nécessite d'être redéfinie dans une classe fille qui aura donc l'objectif d'implémenter les spécifications de la classe abstraite mère.

```
abstract class ClassAbstraite {

        …

        abstract typeretour methodeAbstraite(parametres) ;

}
```

Modificateurs et visibilité

Remarque:

Vue la notion d'abstraction, une méthode abstraite ne peut pas être:

- **synchronized**: car elle n'est pas encore défini.
- **static** : pour la même raison
- **final** : car cela va empêcher sa redéfinition dans la classe fille, or une méthode abstract n'a pas d'existence que lorsqu'elle est définie.
- **private** : la redéfinition d'une méthode privée signifie la définition d'une nouvelle méthode qui porte le même nom. De ce fait on peut pas donner d'existence à une méthode abstraite déclarée private. Ce qui interdit la combinaison **private- abstract**.

La méthode main

La remarque que nous avons fait à propos des méthodes static reste valable pour le main qui est une méthode static. Cela impose une grande restriction, car la méthode main ne peut accéder qu'à des données et des méthodes static. Pour résoudre ce problème, il suffit de créer un objet de la classe principale dans la méthode main. Ce qui implique l'exécution du constructeur. Ce dernier n'est pas une méthode static on peut alors y mettre tous les accès nécessaires aux différents membres non-static. Le squelette d'un programme Java est souvent alors formé de la manière suivante :

La méthode main

```
class NomDeClasse {

        // données membres

        NomDeClasse ()   // Constructeur
        {
                ...
        }

        ...

        public static void main(String args[])
        {
                new NomDeClasse();
        }
}
```

La méthode main

Exemple:

```
class Class1 {
	static int x = 20;
	int y = 30;

	Class1() {
		x = x + 1 ; // correcte
		y = y * 2 ; // correcte
		p2() ; // correcte
	}

	void p2(){
		System.out.println("x = " + x + "     y = " + y) ;
	}

	public static void main(String args[]) {
		new Class1();
		y = y * 2 ; // incorrecte
		p2() ; // incorrecte
	}
}
```