Cours de base d'Ingéniérie des applications objet. Généralités sur le langage Java - Modèle d'exécution - Syntaxe

Support de Cours

Christophe Dony

Université Montpellier-II

1 Le langage Java

1.1 Langage de programmation par objets

Langage de programmation Orienté Objet (types abstraits, surcharge, polymorphisme d'inclusion, liaison dynamique)

Synthèse de Smalltalk (pas de variables de type pointeur, allocation

dynamique, recupération dynamique, machine virtuelle) et de C++ (Syntaxe, constructeurs, ...). C'est un C++ adapté pour plus de simplicité d'utilisation.

1.2 Bibliotheques

La machine virtuelle (l'interpréteur) est livré avec un ensemble de bibliothèques regroupées logiquement en API.

API (Application Programming Interface) : l'interface à une bibliothèque spécialisée dans un certain domaine d'application. Ex API JDBC (Java Database Connectivity).

L'implantation d'une API regroupe un où plusieurs *packages* (cf. Chapître 3). dans lesquels sont définis un ensemble classes et de méthodes.

Ex de packages:

- java.lang regroupe les classes de base necessaire à la définition de tout programme Java (Object, Exception, String, Thread).

- java.math ...

- java.net : classes pour la réalisation d'applications réseau.

- java.util : Classes utilitaires pour la réalisation d'applications classiques (structures de données (Vecteur, Dictionnaire ...), Date, Time, Calendar, Evènements, ...

- java.io : Entrées-Sorties, Serialisation, ...

1.3 Principe d'exécution : Machine virtuelle - Portabilité - Mobilité

Le code compilé n'est pas du code directement exécutable mais un ensemble d'instruction interprétables par la machine virtuelle Java. Java peut être exécuté sous quasi tous les unix, sous windows, sous macOS, ou dans des systèmes embarqués possédant une puce javaOS.

Definition : Machine virtuelle : machine logicielle définie par un ensemble d'instruction et une manière d'interpréter chaque instruction.

Principe d'exécution:

1. compilation des instruction en byteCodes ou instructions de la *machine virtuelle* Java.
2. interprétation de ces instructions par un interpréteur
3. un interpréteur plus l'API définissant les classes de base du système constituent une plate-forme d'exécution Java (JRE : Java runtime environment).

Définition : Interpréteur Un Interpréteur d'un langage est un programme capable de lire les instructions du langage, de les interpréter i.e. de mettre en oeuvre les calculs spécifiés par ces instructions et de faire connaitre, lorsque nécessaire les résultats fournis par ces calculs.

Java est relativement peu gourmand en mémoire. On peut intégrer une machine virtuelle java dans des applications embarquées (un oscilloscope, une montre) ou dans des logiciels (netscape).

Mobilité : Les programmes Java peuvent voyager (applets).

Le code est testé plusieurs fois avant d'être exécuté.

Vérificateur de Byte-Code : teste le format des fragments de code et applique un algorithme qui détermine leur légalité (pas de modification des pointeurs, pas de violation des droits d'accès aux objets).

Class Loader (chargeur de classes). Les classes peuvent être locales (liées à la machine virtuelle en cours d'utilisation) ou chargées via le réseau (Applets). Le chargeur sépare les espaces-noms des deux types de classes (aucune classe importée ne peut se faire passer pour une classe locale, ... en principe).

2 Syntaxe et Eléments de base du langage

2.1 Anatomie des programmes

Un programme est un ensemble de classes dont le texte est défini dans des fichiers (une seule classe publique par fichier). Chaque classe est définie dans une unité de visibilité nommée package.

Contenu (extrait) du fichier *Point.java*

```
package picture;
import java.lang.Math;

public class Point {
    private int x;
private int y;

    public int gety() {return y;}

    public Point setx(int i) {
x = i;
return this;}
```

2.2 Anatomie d'une application Java

Une application indépendante possède un point d'entrée unique qui une classe détentrice d'une méthode nommée *main*. Elle est exécutée directement par la machine virtuelle Java.

```
/**
* The HelloWorldApp class implements an application that
* simply displays "Hello World!" to the standard output.
*/
class HelloWorldApp {
  public static void main(String[] args) {
  System.out.println("Hello World!"); //Display the string.
  }}
```

Compilation : *javac HelloWorldApp.java*, produit *HelloWorldApp.class*

Exécution : *java HelloWorldApp*

2.3 Anatomie d'une applet

Une applet est une application s'exécutant dans un *Breowser*.

Une application de type Applet ne possède pas de point d'entrée *main* mais de points d'entrée sous le contrôle de l'afficheur d'Applets (AppletViewer, Netscape).

Une applet est modélisée par une *sous-classe* de *Applet* implémentant au moins une des 3 méthodes suivantes : *init*, *start*, ou *paint*.

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
  public void paint(Graphics g) {
  g.drawString("Hello world!", 50, 25);
  }}
```

Compilation : *javac HelloWorldApp.java*

Invocation. Dans une page HTML :

```
<APPLET CODE=HelloWorld.class WIDTH=anInt HEIGHT=anInt>
        </APPLET>
```

2.4 Instructions

Sortes d'instructions : déclarations, expressions et structures de contrôles.

expression : série d'opérateurs et d'opérandes syntaxiquement correcte et ayant une valeur.

Identificateur : mot commençant par une lettre.

Règles :

Une affectation est une expression.

Toutes les expressions ne peuvent être utilisées en position d'instruction, le peuvent : affectations, envois de messages, création d'objets.

Les déclarations et les structures de contrôle ne sont pas des expressions (ne rendent pas de valeur).

Terminateur d'instruction: `;`.

2.5 Commentaires

```
// commentaire sur une seule ligne

/* commentaire sur plusieurs
   lignes */

/** commentaire traité
*   par javadoc
    */
```

Javadoc : Utilitaire capable de générer automatiquement de la documentation au format HTML à partir des fichiers sources java.

Tag	Rôle
@author	auteur de l'élement
@deprecated	un élément est déprécié
@docRoot	répertoire principal de génération de la documentation
@exception	permet de préciser une exception qui peut être levée par l'élément
@link	inserer un lien vers un élément de la documentation dans n'importe quel te
@param	permet de préciser un paramètre de l'élément
@see	permet de préciser un élément en relation avec l'élément documenté
@serial	
@serialData	
@serialField	
@since	permet de préciser depuis quel version l'élément a été ajouté 1.0 package, c
@throws	identique à @exception
@version	permet de préciser le numéro de version de l'élément
@return	permet de préciser la valeur de retour d'un élément

2.6 Structures de contrôle

Statement	Keyword
decision making	if-else, switch-case
loop	for, while, do-while
exception	try-catch-finally, throw
miscellaneous	break, continue, label: , return

Exemple while :

```
int count = 0;
while (in.read() != -1) count++;
```

Exemple switch :

```
int month;
. . .
switch (month) {
    case 1:  System.out.println("January"); break;
```

```
    case 2:  System.out.println("February"); break;
    . . .
    default: System.out.println("not a valid month!"); break;
}
```

2.7 Opérateurs

- Arithmétiques binaires : `+, -, *, /, %`
- Affectation : `=, op=, ++, --`

  `{\tt ++a}  équivaut (valeur) à :  {a = a + 1;}`

  `{\tt a++}  équivaut (valeur) à :  {\tt b = a; a = a + 1; b}`

  `{\tt int i = 1; i += 2;}  équivaut à : {\tt int i = 1; i = i + 2;}`
- relationnels : `>, >=, <, <=, ==, !=`
- conditionnels : `&&, ||, !, &. |`

- décalages : `>>`, `<<`

- logiques : `&(AND), |(OR), ^(XOR)`
- déclaration et accès aux tableaux : `[]`,
- déclaration de paramètres : `(params)`,
- blocs : `{}`
- création d'objets : `new`,
- conversion de type : `( type )`
- test de classe d'un objet : `instanceof`
- formation de noms qualifiés : `.`

2.7.1 précédence et associativité

Associativité :

- à précédence égale, tous les opérateurs binaires (sauf ceux d'affectation) associent de gauche à droite.

- affectation : `a = b = c` $\equiv$ `a = (b = c)`

Précédence :

```
operateurs postfixes ([], (params), ++, --) >
unary ops >
creation or cast >
multiplicatifs >
additifs >
décalage >
relationnels >
tests égalité >
& (bitwise AND) >
^ (XOR) >
| (OR) >
opérateurs logiques >
opérateurs d'affectation.
```

2.8 Types

Langage statiquement typé : toute variable et toute expression ont un type.

Les types de java sont les types dits “prédéfinis” et les types dits “référence”. Toute classe définit un nouvel élément de l’ensemble des types “référence”.

2.8.1 Types prédéfinis

byte	8-bit two’s complement	Byte-length integer
short	16-bit two’s complement	Short integer
int	32-bit two’s complement	Integer
long	64-bit two’s complement	Long integer
float	32-bit IEEE 754	Single-precision floating point
double	64-bit IEEE 754	Double-precision floating point
char	16-bit Unicode character	A single character

`boolean  A boolean value (true or false)`

Les caractères sont codés sur 16 bits (unicode caractères) afin d'autoriser l'utilisation de tous les caractères internationaux (ascii, latin, grecs, russes, ...).

2.8.2 A propos des types prédéfinis

Chaque type prédéfini possède des valeurs constantes appelées constantes littérales ou littéraux.

type	littéral
type référence	#null
boolean	true, false
int	035 octal
	0x35 hexa
	35 décimal

2.8.3 Les tableaux

Un tableau est une collection d'entités toutes du même type s'utilisant de façon usuelle (issue de C et C++). Le type tableau est un type référence mais ceci est implanté de façon ad.hoc.

```
int[] tabint; //déclaration d'un tableau d'entiers
tabint = new int[10]; //initialisation de la variable tabint
```

Initialisation des éléments du tableau

```
for int i = 0; i < tab.length; i = i + 1)
{tabint[i] = 0;}
```

accès aux éléments

```
int j = tab[5];
tab[3] = 12;
```

2.8.4 Conversion de types

Le compilateur peut être amené à réaliser des conversions de type et le programmeur peut être amené à en programmer.

– **Conversions implicites** :

Toute valeur numérique d'un type primitif T' peut être affectée à une variable d'un type numérique T si le domaine de valeurs de T est plus large que celui de T'. ($double > float > long > int > short > byte$), ($int > char$).

```
float f = 1234;
int i = 'a';       // i = 97
```

– **Conversions explicites** :

D'autres conversions peuvent être réalisées **explicitement**, avec des risques de perte d'information ou de signalement d'exceptions.

Pour les types primitifs, voir la documentation pour la liste de toutes les possibilités et tous les détails. Un exemple :

```
double d = 7.99;
long l = (long)d;
```

Pour les types référence, voir plus loin dans le cours.

```
Point p = new Point3D(1,2,3);
Point3D p3 = (Point3D) p;
```

2.9 Les Variables

Les variables Java ont un nom et un type.

Une variable peut être initialisée à sa déclaration : `int i = 12;`

Le mot clé *final* transforme, à sa première affectation, une variable en constante nommée : `final int i = 0;`

Par ailleurs, les variables Java ont une portée (*scope*) lexicale.

3 Note

En plus des divers ouvrages publiés consultés pour réaliser ces notes, j'ai également utilisé : "Programmation Java" par Michel Buffa, Peter Sander, Richard Grin, Jean-Philippe Orsini de l'Université de Nice".