La Programmation en Java

RESUME

En programmation, tout ce que nous faisons est un cas particulier de quelque chose de plus général - et souvent nous nous en apercevons trop vite.

Sameh Afi

I. Introduction

I.1. Petit historique

1991 Naissance de Java chez Sun Microsystems.

1995 Réalisation du logiciel HotJava, un navigateur Web écrit par Sun en Java.

Les autres navigateurs Web ont suivi, ce qui a contribué à l'essor du langage sous forme de versions successives :

- 1996 : Apparition du premier JDK sur le site de Sun Microsystems
- 1998 : Lancement de Java 2 avec le JDK 1.2
- 2002 : J2SE 1.4 (J2SE : Java 2 Standard Edition)
- 2004 : J2SE 5.0
- 2006 : Java SE 6

I.2. Caractéristiques

I.2.1. Java : un langage de programmation orientée objet

Objet : ensemble de **données** et de **méthodes** agissant exclusivement sur les données de l'objet.

Encapsulation des données : accès aux données d'un objet par ses méthodes, qui jouent ainsi le rôle d'interface obligatoire. On dit que l'appel d'une méthode d'un objet correspond à l'envoi d'un **message** à l'objet.

Pas d'accès à l'implémentation d'un objet.

Facilite la maintenance et la réutilisation d'un objet.

Classe : description d'un ensemble d'objets ayant une structure de données commune et disposant des mêmes méthodes. Un objet est une **instance** de sa classe.

Héritage : permet de définir une nouvelle classe à partir d'une classe existante, à laquelle on ajoute de nouvelles données et de nouvelles méthodes.

Facilite la réutilisation de classes existantes.

Certains langages, tels C++, offrent la possibilité d'un héritage multiple (une même classe peut hériter simultanément de plusieurs autres). Ce n'est pas le cas de Java.

I.2.2. Java et la portabilité

Dans la plupart des langages de programmation, un programme **portable** est un programme dont le code peut être exploité dans différents environnements moyennant simplement une nouvelle compilation.

En Java, la portabilité va plus loin…

En java, un programme est à la fois compilé et interprété.

La compilation d'un programme Java produit un code intermédiaire exécutable formé de **bytecodes**, qui est un pseudo code universel disposant des fonctionnalités communes à toutes les machines.

L'interpréteur Java appelé **machine virtuelle** (JVM pour Java Virtual Machine) permet d'exécuter le bytecode produit par le compilateur.

I.2.3. Applets et applications autonomes

A l'origine Java a été conçue pour réaliser des applets s'exécutant sous le contrôle d'un navigateur Web. Les applets sont un flop, réorientation vers les applications serveur avec J2EE (Java 2 Enterprise Edition) en 1999.

Java permet d'écrire des programmes indépendants du Web. On parle alors d'**applications autonomes**. Une application autonome s'exécute sous le contrôle direct du système d'exploitation. Il n'y a pas besoin de navigateur Web pour l'exécuter.

II. La syntaxe du langage Java

II.1.Un premier programme

```
public class PremProg
{
   public static void main(String args[])
   {
      int i, total=0, n=5;
      for(i=1;i<=n;i++)
      total=total+i;
      System.out.println("somme des entiers de 1 a " + n + " = " + total);
   }
}
```

Une application peut être formée d'une seule classe, la classe PremProg.
Le mot clé public dans public class PremProg sert à définir les droits d'accès des autres classes (en fait de leurs méthodes) à la classe PremProg.
La méthode main est une méthode particulière qui représente le code à exécuter.
System.out.println correspond à une fonction d'affichage dans la fenêtre console.
Le résultat de l'exécution de ce programme est :

```
somme des entiers de 1 a 5 = 15
```

Java dispose de trois formes de commentaires :

- Commentaires usuels placés entre les délimiteurs /* et */ ;
- Commentaires de fin de ligne introduits par le délimiteur // ; Commentaires de documentation entre les délimiteurs /** et */ : leur intérêt est de pouvoir être extraits automatiquement par des programmes utilitaires de création de documentation tels que Javadoc (création d'une aide en ligne au format HTML à partir de fichiers source Java correctement documentés).

II.2.Les types primitifs

II.2.1. Les entiers relatifs

Les différents types

type	taille	domaine de valeurs
byte	8 bits	-128 à 127
short	16 bits	-32 768 à 32 767
int	32 bits	-2 147 483 648 à 2 147 483 647
long	64 bits	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

II.2.2. Les types flottants

type	taille	précision (chiffres significatifs)	domaine de valeurs (pour les valeurs positives)
float	32 bits	7	1.40239846E-45 à 3.40282347E38
double	64 bits	15	4.9406564584124654E-324 à 1.797693134862316 E308

II.2.3. Le type caractère

Les caractères sont de type char et sont représentés en mémoire sur 16 bits en utilisant le code universel Unicode.
Notations des constantes de type caractère : classique entre apostrophes (Exemple : 'a', 'b', 'é')

II.2.4. Le type booléen

Le type boolean permet de représenter une valeur logique de type vrai/faux. Les deux constantes du type boolean se notent true et false.

II.2.5. Les variables de type primitif

Une **définition de variable** précise le type, l'identificateur et éventuellement une ou plusieurs valeurs d'initialisation
type identificateur [= valeur(s) d'initialisation]

Un **identificateur** est une suite de caractères formés avec des lettres, des chiffres ainsi que les caractères _ (souligné) et $. Le premier caractère ne peut pas être un chiffre. Les minuscules sont distinguées des majuscules.

La définition d'une variable peut apparaître à n'importe quel emplacement du programme. Mais, **toute variable doit être initialisée avant d'être utilisée**.

Exemple

```
int n=10,m,l;
int p=2*n;                                  //ok
l=30;                                       //ok
System.out.println("m = " + m);
    //erreur de compilation : la valeur de m n'est pas définie
```

II.2.6. Instructions, portée d'une variable

Une **instruction** en Java est :
Une **instruction simple** toujours terminée par un point-virgule et pouvant être librement écrite sur plusieurs lignes ;

Une **instruction composée**, encore appelée **bloc d'instructions**, qui est une suite d'instructions placées entre accolades.

La **portée** d'une variable va de la ligne de sa définition jusqu'à la fin du bloc dans lequel elle est définie.

Exemple

```
{
    int          n=10;
    …
    {
        int p=20;
        float n=3.5;
            //erreur de compilation : définition de n invalide.
        …
    }
    System.out.println("p = " + p);
       //erreur de compilation : p inconnu
}
```

II.2.7. Les constantes

Le mot clé **final** permet de déclarer que la valeur d'une variable ne doit pas être modifiée pendant l'exécution du programme.

Exemple

```
final      int      n=10,m;
n++;
    //erreur de compilation : n a été déclaré final
m=n*n;
    //ok bien que la valeur de m ne soit connue qu'à l'exécution
int p=2*m;
```

II.3.Les opérateurs et les conversions

II.3.1. Les opérateurs arithmétiques

Les types de base byte, short, int, long, float, double et char admettent comme opérateurs : **+ - * /** dont les priorités et l'associativité sont celles des autres langages.

Le quotient de deux entiers est un entier (5/2=2), alors que le quotient de deux flottants est un flottant (5.0/2.0=2.5).

L'opérateur **modulo** noté **%** peut porter sur des entiers ou des flottants. Il fournit le reste de la division entière de son premier opérande par son second.

Exemple : 11%4 = 3; 12.5%3.5 = 3

La division par 0 pour les entiers conduit à une erreur d'exécution. Par contre, aucune opération sur les flottants ne conduit à un arrêt d'exécution.

II.3.2. Les conversions implicites de types

Il existe deux conversions implicites de types : les conversions d'ajustement de type et les promotions numériques.

Les **conversions d'ajustement de type** (int => long => float **=>** double) sont automatiquement mises en œuvre par le compilateur, à condition qu'il n'y ait pas rétrécissement du domaine.

Exemple

```
int n; long p; float x;
n*p+x;
```

conversion d'ajustement de n en long, multiplication par p ; Le résultat de n*p est de type long, conversion d'ajustement en float pour être additionné à x ; le résultat final est de type float.

Les **promotions numériques** convertissent automatiquement toutes valeurs de type byte, short ou char apparaissant dans une opération arithmétique en int.

Exemple

```
short n,p; float x;
n*p+x;
```

promotions numériques de n et de p en int et multiplication ; Conversion d'ajustement du résultat de n*p en float pour être additionné à x ; Le résultat final est de type float.

II.3.3. L'opérateur d'affectation

L'affectation renvoie la valeur de la variable affectée.

Cet opérateur possède une associativité de droite à gauche. On peut ainsi écrire des affectations en cascade.

Exemple : i=j=k=5 ;

Les **conversions implicites par affectation** sont se font suivant l'une des deux hiérarchies suivantes :

byte **=>** short => int => long => float => double

char => int => long => float => double

Vous pouvez **affecter** n'importe quelle **expression constante entière** à une variable de type byte, short ou char, à condition que sa valeur soit représentable dans le type voulu.

Exemple

```
final int N = 50 ;
short p = 10 ;
char c = 2*N + 3 ;          //ok : c est le caractère de code 103
byte b = 10*N;              //erreur de compilation : 500 est supérieur à la
                            capacité du type byte
```

II.3.4. Les autres opérateurs

Opérateurs relationnels <, <=, >, >=, == et **!=** dont le résultat est de type boolean.

Pour les caractères, on a :

'0'<'1'<…<'9'<'A'<'B'<…<'Z'<'a'<'b'…<'z'

Opérateurs logiques dont voici la liste classée par priorité décroissante. Le résultat est de type boolean.

Opérateurs	Signification	Valeur de vérité	Remarques
!	non	!cond : vrai si la condition est fausse	
&	et	cond1 & cond2 : vrai si les deux conditions sont vraies	les deux opérandes cond1 et cond2 sont évalués.
|	ou	cond1 | cond2 : vrai si **au moins une** des deux conditions est vraie	les deux opérandes cond1 et cond2 sont évalués.
^	ou exclusif	cond1 ^ cond2 : vrai si **une et une seule** des deux conditions est vraie	les deux opérandes cond1 et cond2 sont évalués.
&&	et conditionnel	comme &	l'évaluation de la condition est finie dès qu'elle devient fausse.
||	ou conditionnel	comme |	l'évaluation de la condition est finie dès qu'elle devient vraie.

Opérateur ternaire ? : qui a le même comportement qu'en C/C++.

Exemple

```
int a=10, b=5, max ;
max = a>b ? a : b ;                    //max vaut 10
```

Opérateurs d'incrémentation et de décrémentation

Les opérateurs **++** et **--** peuvent être postfixés ou préfixés.

Exemple

```
int i = 3 ;
int j = i++ ;                  //j vaut 3 et i vaut 4
int k = ++i ;                  //k et i valent 5
```

Les opérateurs d'incrémentation n'appliquent pas de conversion à leur opérande.

Exemple

```
byte i ;
i=i+1 ;          //erreur de compilation : i+1 de type int
                 ne peut pas être affecté à un byte
i++ ;            //ok
```

II.3.5. Les conversions explicites de types : l'opérateur cast

Le programmeur peut forcer la conversion d'une expression quelconque dans un type de son choix, à l'aide de l'opérateur **cast**.

Exemple

```
short x=5, y=15 ;
x = (short) (x+y);             //ok grâce au cast
```

L'opérateur **cast** a une grande priorité. Exemple

```
short x=5, y ;
y = (short) x+2;               //erreur de compilation :
                    conversion de x en short, promotion numérique de x en int ; le résultat
                    de l'addition est de type int et ne peut pas être affecté à un short.
```

Le résultat d'une conversion explicite par un **cast** peut conduire à un résultat fantaisiste.

Exemple

```
int n=500 ;
byte b=(byte) n ;          //légal mais b aura pour valeur -24,
```

500 étant supérieur à la capacité du type byte

II.3.6. Récapitulatif des priorités des opérateurs

Liste des opérateurs classés par ordre de priorité décroissante et accompagnés de leur mode d'associativité.

Opérateurs	**Associativité**
[] ()	gauche à droite
++ -- ! cast new	droite à gauche
* / %	gauche à droite
+ -	gauche à droite
< <= > >= instanceof	gauche à droite
== !=	gauche à droite
&	gauche à droite
|	gauche à droite
&&	gauche à droite
||	gauche à droite
? :	gauche à droite
=	droite à gauche

L'opérateur **.** permet de faire référence à un champ ou à une méthode d'un objet.
L'opérateur **[]** permet de faire référence à un élément d'un tableau.
L'opérateur **()** permet d'appeler une méthode.
L'opérateur **instanceof** permet de tester si un objet est l'instance d'une classe donnée.

II.4.Les instructions de contrôle

II.4.1. Enoncés conditionnels

if (condition**)**
    instruction_1
[**else**
    instruction_2]

condition est une expression booléenne.
instruction_1 et instruction_2 sont des instructions quelconques.

Les instructions **if** peuvent être imbriquées.

switch (expression**)**
{ case constante_1 **:** [suite_d'instructions_1]
case constante_2 **:** [suite_d'instructions_2]
…
case constante_n **:** [suite_d'instructions_n] [**default :** suite_d'instructions]
}

expression est une expression de l'un des types byte, short, char ou int.
constante_i est une expression constante d'un type compatible par affectation avec le type de expression.
suite_d'instructions_i est quelconques.

Exemple

```
char c ;
switch (c)
{ case 48 : System.out.println(c) ; break ;
    case 'a' :      System.out.println("c est la lettre a") ;
                    break           ;
 }
```