Programmation Orientée Objet

Mohamed Tounsi

Institut Supérieur d'Informatique et de Multimédia Sfax

Septembre 2014

Syntaxe de base

Les premières règles à retenir sont les suivantes:

- Chaque instruction se termine par un point-virgule, c'est absolument obligatoire.
- Le langage est case-sensitif (sensible à la casse).
- Les instructions peuvent être regroupées dans des blocs d'instructions, ils se délimitent par des accolades: { et }.
- Toute variable utilisée doit être au préalable déclarée.
- Toute variable déclarée doit être typée.
- Les commentaires sur une ligne seront précédés d'un //.
- Les commentaires sur plusieurs lignes seront encadrés par /* au début et par */ à la fin.

Les types de base (1)

1. L'utilisation d'une variable nécessite obligatoirement de la déclarer.
2. La déclaration d'une variable se fera en spécifiant son type, son nom et éventuellement une valeur d'initialisation.
3. *String* est le type chaîne de caractères. Souvent considéré comme un type primitif, il s'agit en fait d'un type objet.

```
String chaine= "Bonjour tout le monde";
```

4. Les variables peuvent être déclarées partout, à tout moment.
5. La visibilité des variables est limitée au bloc d'instruction dans lequel elles sont déclarées, et à tous les blocs qui y sont inclus.

Les types de base (2)

Type	Signification	Taille (bits)
byte	Entiers signés	8
short	Entiers signés	16
int	Entiers signés	32
long	Entiers signés	64
float	Virgule flottante	32
double	Virgule flottante	64
char	Non signés	16
boolean	True ou false	1

Les types primitifs de Java

Les opérateurs
Opérateurs arithmétiques

Définition

Ils permettent toutes les opérations arithmétiques sur des variables entières ou à virgule flottante. Ils retournent une valeur d'un type identique à l'opérande de gauche.

Syntaxe

```
+
-
*
/
%
```

Exemple

```
String s="Bonjour" + "Isims";
int n1=5/2;
int n2=5\%2;
```

Remarque

L'opérateur + peut s'appliquer aux chaînes de caractères (String). C'est le seul opérateur arithmétique qui peut s'appliquer à ce type de données. Il prend alors la fonction d'opérateur de concaténation.

Les opérateurs
Opérateurs d'affectation

Définition

Ils permettent d'affecter une valeur à une variable. L'opérateur "=" couplé avec un opérateur arithmétique permet de retourner dans l'argument de gauche l'opération arithmétique de celui-ci avec l'argument de droite.

Syntaxe

```
=
+=
=+
-=
=-
*=
=*
etc....
```

Exemple

```
c+=8; //est équivalent à c=c+8 ?
a+=3; //fait la même chose que a=+3 ?
```

c=a+=3; et c=a=+3; Sont-ils identiques?

Les opérateurs
Opérateurs unaires

Définition

Ils permettent d'incrémenter ou de décrémenter leur unique opérande.

Syntaxe

```
++
--
```

Exemple

```
int i= 4; int j=4;
System.out.println("i++= "+i++ +", ++j= "+ ++j);
```

ce qui donne:

```
i++= 4, ++j=5
```

Dans le premier cas, la valeur de l'opérande est retournée avant qu'elle ne soit incrémentée, dans le second cas après.

Les opérateurs
Opérateurs bit à bit

Définition

Ils permettent d'effectuer des opérations booléennes sur chaque bit des variables. Les variables doivent obligatoirement être de types entiers.

Syntaxe

```
&   opérateur AND (ET)
|   opérateur OR (OU)
~   opérateur NOT (NON)
>> décalage à droite
<< décalage à gauche
```

Exemple

```
System.out.println ("34 & 24 = "+(34 & 24));
System.out.println ("34 | 24 = "+(34 | 24));
System.out.println ("34 & ~24 = "+(34 & ~24));
System.out.println ("34 >> 2 = "+(34 >> 2));
System.out.println ("34 << 2 = "+(34 << 2));
```

Les opérateurs

Opérateurs de comparaisons logiques

Définition

Ces opérateurs prennent en argument des variables de types quelconques, mais retournent systématiquement une valeur booléenne, donc *true* ou *false*.

Syntaxe

```
==
!=
<
<=
>
>=
```

Exemple

```
System.out.println ("34 > 12: "+ (34>12) +",
25 != 25: "+ (25!=25));
```

Résultat:

```
34 > 12: true,  25 != 25: false
```

Les opérateurs

Opérateurs logiques sur des booléens

Définition

Les opérandes de ces opérateurs logiques sont exclusivement des booléens. Le résultat est aussi un booléen.

Syntaxe

```
!   opérateur AND
&&  opérateur OR
||  opérateur NOT
```

Exemple

```
boolean b1= false; boolean b2=false;
System.out.println("False ou True: "+(false || true)
+", !b1 && !b2: "+(!b1 && !b2));
```

Résultat:

```
False ou True: true, !b1 && !b2: true
```

Les opérateurs
Opérateur ternaire

Définition

Bien connu en C et C++, il permet en une seule instruction de rendre un résultat conditionné par une relation booléenne.

Syntaxe

```
?:
```

Exemple

```
int g= 3; int h= 6;
int x=g>h ? g:h;
```

Dans cet exemple, la variable x recevra comme valeur maximum entre g et h (ce sera donc la valeur de h, soit 6).

Transtypage

définition

Le Transtypage permet de faire passer le contenu d'une variable d'un type dans une variable d'un autre type. Il faut toutefois que ces types soient compatibles (les entiers et les flottants ne le sont pas).

Deux méthodes sont possibles:

- **Implicite**: Lorsque on fait entrer le contenu d'une variable d'un type vers un type plus grand (int dans long)
- **Explicite (cast)**: Dans ce cas, on précise le type vers lequel convertir la donnée en mettant son nom entre parenthèse devant la varaible à convertir.

```
long x;
// on donne une valeur à x
int i= (int) x;
```

La structure conditionnelle

if... else...

Définition

Cette instruction permet d'effectuer un traitement parmi deux, en fonction de l'évaluation d'une expression booléenne passée en argument du *if*.

Syntaxe

```
if (expression_booléenne)
 instruction(s)_si_vrai;
 else
 instruction(s)_si_faux;
```

Exemple

```
int a=1;
if(a==0)
  if(true)
    System.out.println("a est à 0");
  else
    System.out.println("on ne passe jamais ici");
else
  System.out.println("a n'est pas à 0);
```

Les boucles
While

Définition

Elle permet la répétition d'une instruction ou d'un bloc d'instructions autant de fois qu'une condition est remplie.

Syntaxe

```
while (expression_booléenne)
{
 instruction(s);
}
```

Exemple

```
int n=0;
while (n<10);
/* Attention!!! ici le point-virgule est un bug classique */
/*                on boucle sans arrêt car                */
/*        il n'y a pas d'opération dans la boucle         */
{  n++; }
```

Les boucles
do...While

Définition

Elle est semblable à la boucle *while*, sauf que dans le *do ..... while* l'instruction ou le bloc d'instruction sera exécuté au moins une fois.

Syntaxe

```
do
{
 instruction(s);
}
while(expression_booléenne);
```

Exemple

```
int n=0;
do
 {
   System.out.println("Je le fais 5 fois ");
   n++;
 }
while (n<5);
```

Les boucles
for

Définition

Le *for* permet des boucles avec instruction d'initialisation. On peut mettre plusieurs instructions initiales et instructions récurrentes dans les arguments du *for*, à condition de les séparer par une virgule.

Syntaxe

```
for(instruction_initiale;
expression_booléenne;
istruction_récurente)
{
 instruction(s);
}
```

Exemple

```
for(int x=0; x<5; x++)//la visibilité de x est limitée à for
System.out.println("x= "+x);

int n=0;
for( ; n<5; )//c'est équivalent à un while
{System.out.println("x= "+x); n++;}

for(;;)//c'est un "forever"
{System.out.println("Boucle sans fin"); return;}// Non!!
```

Le switch

Définition

Le *switch* ne s'applique qu'a des valeurs entières ou à des char. Chaque cas se termine en général par un *break*; qui permet de sortir du *switch*. Si on omet le *break*, alors les instructions du *case* suivant seront exécutées, et ainsi de suite jusqu'au premier *break* rencontré ou jusqu'à la fin du *switch*.

Syntaxe

```
switch(expression Entière/Char)
{
  case valeur_1: instruction;
       break;
  case valeur_1: instruction;
       break;
  //etc....
  default:  instructions;
}
```

Exemple

```
int saison= 1; String nom_saison;
switch (saison){
case '1': nom_saison ="ETE"; break;
case '2': nom_saison ="AUTOMNE"; break;
case '3': nom_saison ="HIVER"; break;
default: nom_saison="PRINTEMPS";}
```

Instructions d'interruption

Les instructions *break* et *continue* sont disponibles dans certaines structures de contrôle.

- *break* permet de sortir d'une boucle,
- *continue* permet de sortir de la boucle, mais en retournant à la condition booléenne.

Exemple:

```
for(int n=0; n<10; n++) {
 if (n==3) continue;
 System.out.println("n= "+n);
 if(n==4) break;
}
```

Résultat:

```
n=0
n=1
n=2
n=4
```

Les tableaux (1)

définition

Un tableau est composé d'un nombre déterminé de variable d'un même type (primitif ou objet). Les cellules non initialisées contiennent 0 (pour des types primitifs) ou null (pour des objets).

- *Déclaration d'un tableau* Type [] nomTableau;
- *Création du tableau* nomTableau = new Type[taille];
- *Initialisation du tableau* nomTableau[numeroCellule] = UneValeur;

Remarque

La taille du tableau est définie au moment de sa création, et ne peut plus être changée par la suite. Si on manque de la place dans un tableau, il faut obligatoirement en créer un nouveau plus grand.

Les tableaux (2)

- L'accès aux cellules se fait en spécifiant un nombre entier.

```
int [] tab;
tab = new int[20];
tab[0]= 15;
```

- L'initialisation d'un tableau peut se faire par une liste statique d'initialisation comme suit:

```
int tab [] = {3, 11, 9, 0, 44, 72, 5}; // 7 cellules
String [] tab2 ={"Hello", "Hola", "Bienvenue"}; // 3 cellules
```

Les tableaux à plusieurs dimensions

définition

Les tableaux multidimensionnels sont simplement des tableaux de tableaux. Le nombre de dimensions est sans limite (attention à la taille de la mémoire).

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

- On peut aussi utiliser des listes statiques d'initialisation pour des tableaux multidimensionnels:

```
int [][] tab3= {{23,45,1,0}, {12,3}, {4,23,5,65}, {2}};
```

- On peut déclarer un tableau composé de tableaux dont les tailles sont différents. C'est tout à fait autorisé.

```
int [][] ti;
ti = new int [];
ti[0] = new int [20];
ti [1] = new int [14];
```

sources

- *Java, la maîtrise , Guide formation avec exercices corrigés* de Jérôme Bougeault - Guide (broché). Paru en 02/2008.
- *Java : La synthèse* de Gilles Clavel , Valérie Lehman, Nicolas Mirouze, Emmanuelle Mouthe, Sandrine Munerot, Emmanuel Pichon, Mohamed Soukal et Simon Tiffanneau. Paru en 9 octobre 2003
- *Exercices en Java* de Claude Delannoy - Guide (broché). Paru en septembre 2011.