Eléments de base du langage JAVA

Un programme en JAVA consiste en une série d'enoncés (statements) organisés en blocs.Ces énoncés définissent des données et des opérations sur ces données. La fin de chaque énoncé est marquée par un ;.Ce ; indique à l'interpréteur JAVA que l'enoncé est complet, qu'il doit être exécuté avant de passer à l'énoncé suivant L'ordre dans lequel l'interpréteur JAVA exécute les énoncés s'appelle le flot d'exécution ou flot de contrôle.

1.Types simples

type	désignation		domaine	constantes		opérations
entier	**byte** **short** **int** **long**	entiers sur 16 bits entiers sur 16 bits entiers sur 32 bits entiers sur 32 bits	−128 à 127 −32768 à 32767 −2147483648 à 2147483647 −4294967296 à 4294967295	323 0xF2 0777	constante décimale constante héxadécimale constante octale	unaire : +, − binaire : +, −, *, / % (modulo)
réel	**float** **double**	réels simple précision (32 bits) réels double précision (64bits)	1.40129846432481707e−45 à 3.40282346638528860e+38 4.94065645841246544e−324 à 1.79769313486231570e+308 MINDOUBLE	2. .32 2.33 2e4 2.e4 .3e4 2.3e4 2.3e−4		unaire : +, − binaire : +, −, *, /
caractères	**char**		valeurs entières sur 16 bits codage Unicode	'a' 'A' '\n' '\t'	retour à la ligne tabulation horizontale	
booléen	**boolean**			true false		&& et logique || ou logique ! négation

Relation d'ordre : elle est définie sur tous les types simples.

égal	différent	strictement supérieur	supérieur ou égal	strictement inférieur	inférieur ou égal
==	!=	>	>=	<	<=

2. Enoncés élémentaires

a) déclaration d'une variable

avant d'être utilisée dans un programme une varaible doit être déclarée

• syntaxe : *déclaration:*
type identificateur

ex :
```
byte age;
boolean  jeune;
float  poids;
```

b) Opérateur d'affectation

permet de donner une valeur à une variable.

• syntaxe : *expression d'affectation:*
identificateur de variable **=** *expression*

ex :
```
age = 25;
jeune = age < 21;
poids = 57.72;
```

Attention en JAVA comme en C, l'affectation est un opérateur et non pas une instruction !! L'affectation peut donc être utilisée comme une expression dont la valeur est la valeur affectée à la variable.

c) Intialiseurs

avant de pouvoir utiliser une variable il faut lui donner une valeur initiale.

ex :
```
int i;
i = i + 1;  illegal i n'est pas initialisé
```

Cela peut être fait avec une affectation.

ex :
```
int i;
i = 0;
i = i + 1;
```

La manière la plus efficace de procéder est d'utiliser un initialiseur, c'est à dire une affectation optionnelle pouvant faire partie d'une déclaration.

ex :
```
int i = 1;
```

3. Flot de contrôle

a) bloc

• syntaxe :

{
séquence d'énoncés
}

• sémantique : permet de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction.

ex :
```
int i = 1;
int j = 12;

{
      j = i -1;
      i = 2 * j + i;
}
```

b) instruction conditionnelle

• syntaxe : **if (** *expression booléene* **)** *énoncé*

ou bien

if (*expression booléenne* **)**
énoncé1
else
énoncé2

• sémantique : *expression* est évaluée. Si la valeur rendue est vraie (`true`), on exécute *énoncé1*, sinon on exécute *énoncé2* si elle existe.

ex :
```
if (i == j) {
      j = i -1;
      i = 2 * j + i;
}
else {
      j = j*2;
      i = i-1;
}
```

e) instructions itératives

Ces structures de contrôle permettent d'exécuter des instructions de manière répétitive.

instruction while (boucle tantque... faire)

• syntaxe :

```
while (expression booléenne)
      enoncé
```

• sémantique :

1) *expression* est évaluée, si sa valeur est vraie (`true`) l'étape 2 est réalisée sinon l'exécution de l'instruction `while` est terminée

2) *énoncé* est exécuté puis l'étape 1 est renouvellée

ex :

```
// calcul de la somme des 10 premiers entiers
int i = 1;
int somme = 0;

while(i <=10)
{
      somme = somme + i ;
      i = i+1;
}

System.out.println("Somme  = "+somme);
```

instruction do-while (boucle faire ... tantque)

• syntaxe :

```
do
      enoncé
while (expression booléene) ;
```

• sémantique :

1) *énoncé* est exécuté

2) *expression* est ensuite évaluée, si sa valeur est vraie (`true`) l'étape 1 est renouvellée sinon l'exécution de l'instruction `do while` est terminée

ex :

```
// calcul de la somme des 10 premiers entiers
int i = 1;
int somme = 0;

do
{
      somme = somme + i ;
      i = i+1;
} while(i <=10);

System.out.println("Somme  = "+somme);
```

instruction for (boucle pour)

• syntaxe :

```
for(expression1; expression2; expression3)
      enoncé
```

• sémantique :

1) *expression1* est évaluée (en général elle initialise l'itérateur, cf l'affectatino qui est une expression). Cette expression n'est évaluée qu'une fois.

2) *expression2* est évaluée, si sa valeur est vraie (`true`) le contrôle passe à l'étape 3 sinon l'exécution de l'instruction `for` est terminée.

3) *énoncé* est exécuté puis *expression3* est évaluée (en général elle modifie l'itérateur) puis l'étape 1 est renouvellée. *expression3* est évaluée après chaque itération.

ex :
```
// calcul de la somme des 10 premiers entiers
int i = 1;
int somme = 0;

for (i = 1; i <= 10; i= i+1)
      somme = somme + i ;

System.out.println("Somme  = "+somme);
```

4 . Affichage de résultats à l'écran

`System.out.print(``chaine à afficher`**`)`** affiche un message sur la console
`System.out.println(``chaine à afficher`**`)`** affiche un message avec retour à la ligne

chaine-à afficher est le texte à imprimer il peut être :

simplement une chaine de caractères ex : `System.out.println("bonjour");`

une expression d'un type simple ex : `System.out.println(age);  // affiche valeur de la variable age`

une combinaison des deux, on utilise alors l'opérateur + (concaténation de chaînes)

ex : `System.out.println("L'age de la personne est " + age + "son poids " + poids);`