CENTRE DE RÉADAPTATION MULHOUSE
Rééducation et Formation Professionnelle

Java

Syntaxe de base

CONTENU

SYNTAXE DE BASE

L'objectif de Java est généralement :

1) Effectuer un traitement (calculs, chargement de fichiers ou de données…).
2) Afficher le résultat du traitement dans le flux de sortie.

/!\ Une instruction Java se termine toujours par un point-virgule **;**

L'instruction **System.out.println("texte à afficher")** permet de déclencher un affichage dans le flux de sortie :

```
public static void main(String[] args)
{
    System.out.println("Hello world!");
}
```

Le texte situé entre les guillemets sera affiché dans le flux de sortie.

Un texte situé entre des guillemets est une **chaine de caractères** ou **String**.

VARIABLES

Une variable est un symbole qui permet de stocker une valeur en mémoire.
Pour faire simple, un symbole est un nom de variable qui est un "raccourci" vers une valeur stockée en mémoire.

En Java, les variables sont en camelCase et possède un type (chaine de caractère, nombre etc…).

```
public static void main(String[] args)
{
    String maVariable; // Déclaration d'une variable de type "String"

    maVariable = "Hello World!"; // affectation d'une valeur à la variable

    System.out.println(maVariable); // affiche "Hello world!"
}
```

Ci-dessus, La valeur "**Hello Wolrd!**" est stockée en mémoire dans la variable "**maVariable**" et il est possible d'y accéder à nouveau en utilisant la variable (le symbole) associée, par exemple pour l'afficher.

POURQUOI "VARIABLE" ?

Comme son nom l'indique, la valeur d'une variable peut… varier dans le temps.
Tout au long de l'exécution du programme, la valeur des variables créées peut changer. Vous comprenez maintenant le nom 😉.

```
public static void main(String[] args)
{
    String maVariable; // Déclaration d'une variable de type "String"

    maVariable = "Hello World!"; // affectation d'une valeur à la variable

    System.out.println(maVariable); // affiche "Hello world!"

    maVariable = "Welcome."; // affectation d'une nouvelle valeur à la variable

    System.out.println(maVariable); // affiche "Welcome."
}
```

TYPES DE VARIABLES

Une variable stocke une valeur d'un certain type. Un type représente le format de la donnée stockée.

Principaux types simples en Java :

Type	Format	Description	Affichage avec System.out.println()
Boolean	Booléen	2 valeurs possibles : **true** ou **false** (vrai ou faux)	**Oui**
int	Nombre entier 32 bits	1 nombre entier positif ou négatif	**Oui**
float	Nombre réel 32 bits	1 nombre à décimales positif ou négatif	**Oui**
string	Chaine de caractères	Du texte, html, xml…	**Oui**
long	Nombre entier 64 bits	1 nombre entier positif ou négatif	**Oui**
double	Nombre réel 64 bis	1 nombre à décimales positif ou négatif	**Oui**

```
public static void main(String[] args)
{
    String chaine = "Une chaine de caractères"; // valeur entre double guillemets

    Boolean vraiFaux = true; // true ou false

    int nombre32bits = 42;

    long nombre64bits = 90123456789L; // L à la fin du nombre

    float nombreAvirgule = 13.0123456789F; // F à la fin du nombre

    double nombreAvirgule64bits = 13.0123456789D; // D à la fin du nombre


    System.out.println(chaine);
    System.out.println(vraiFaux);
    System.out.println(nombre32bits);
    System.out.println(nombre64bits);
    System.out.println(nombreAvirgule);
    System.out.println(nombreAvirgule64bits);
}
```

Java est un langage dit **fortement typé**, le type de chaque variable est précisé à la déclaration.

Lorsqu'une variable est déclarée, elle ne peut contenir que des valeurs correspondant à son type.

CHAINES DE CARACTERES

Dans une application Console, et après avoir effectué ses différents traitements, un programme Java affichera généralement le contenu de chaines de caractères.

```
public static void main(String[] args)
{
    System.out.println("Hello world!");
}
```

CONCATENATION

La concaténation désigne l'action de mettre bout à bout plusieurs chaines de caractères.
Le principe est de séparer les différentes chaines de caractères par l'opérateur de concaténation.

En Java, l'opérateur de concaténation est le caractère plus "**+**".

```
public static void main(String[] args)
{
    String chaine = "Bonjour";
    String chaine2 = "à vous !";

    System.out.println(chaine + " et bienvenue " + chaine2);
}
```

Affiche "Bonjour et bienvenue à vous !"

LES EXPRESSIONS

Une expression est un segment de code effectuant un traitement et pouvant retourner une valeur.

```
int resultat = 3 + 2;
```

Exemple d'expression simple

EXPRESSION BOOLEENNE

Une expression booléenne est une expression dont le résultat sera… une valeur booléenne (true ou false).
Une expression booléenne utilise les **opérateurs de comparaison** et les **opérateurs logiques**.

Opérateurs de comparaison :

Symbole	Nom	Exemple	Signification
==	est égal à	`a == b`	La valeur de a est-elle égale à la valeur de b ?
!=	est différent de	`a != b`	La valeur de a est-elle différente de la valeur de b ?
<	est strictement inférieur à	`a < b`	La valeur de a est-elle strictement inférieure à la valeur de b ?
<=	est inférieur ou égal à	`a <= b`	La valeur de a est-elle inférieure ou égale à la valeur de b ?
>	est strictement supérieur à	`a > b`	La valeur de a est-elle strictement supérieure à la valeur de b ?
>=	est supérieur ou égal à	`a >= b`	La valeur de a est-elle supérieure ou égale à la valeur de b ?

Opérateurs logiques :

Symbole	Nom	Exemple	Signification
&&	ET	`(a == b) && (a > c)`	La valeur de **a** est-elle égale à la valeur de **b** ? **ET** La valeur de **a** est-elle supérieure à la valeur de **c** ?
||	OU	`(a == b) || (a < c)`	La valeur de **a** est-elle égale à la valeur de **b** ? **OU** La valeur de **a** est-elle inférieure à la valeur de **c** ?

Exemples d'expressions booléennes

```
int a = 1;
int b = 2;
int c = 5;

Boolean test1 = (a < b);  // vrai
Boolean test2 = (a < b) && (a > c);  // faux
Boolean test3 = (a < b) || (a > c);  // vrai
```

BLOC DE CODE

Un bloc de code est un ensemble d'instructions rassemblées entre accolades **{ }**.
Un bloc de code peut être considéré comme une expression complexe (contenant plusieurs instructions).

```
7       {
8           int a = 1;
9           int b = 2;
10          int c = 5;
11      }
12
13      Boolean test1 = (a < b);  // vrai
14      Boolean test2 = (a < b) && (a > c);  // faux
15      Boolean test3 = (a < b) || (a > c);  // vrai
```

Les variables déclarées ne sont disponibles que dans le bloc de code dans lequel elles sont déclarées.

Dans la capture précédente, les variables a, b et c (lignes 8, 9 et 10) ne sont pas accessibles en dehors de leur bloc de code délimité par les accolades dans lesquelles elles se trouvent (lignes 7 et 11). C'est pour cette raison que l'éditeur de code les souligne en rouge (lignes 13, 14 et 15). Tenter d'exécuter ce programme résultera une erreur de compilation.

Les blocs de code sont généralement utilisés pour définir le code à exécuter après une instruction conditionnelle, une boucle ou lors de l'appel d'une fonction.

LES STRUCTURES CONDITIONNELLES

Une structure conditionnelle permet de définir quel bloc de code exécuter selon le résultat d'une expression booléenne.

IF / ELSE

```java
int a = 1;
int b = 2;

/** Si a est supérieur à b
 *  ALORS
 *      ECRIRE a, " est supérieur à ", b
 *  SINON
 *      ECRIRE a, " est inférieur à ", b
 *  FIN SI
 */
if(a > b) {
    System.out.println(a + "est supérieur à " + b);
}
else {
    System.out.println(a + "est inférieur à " + b);
}
```

IF / ELSEIF / ELSE

```java
int a = 1;
int b = 2;

/** Si a est supérieur à b
 *  ALORS
 *      ECRIRE a, " est supérieur à ", b
 *  SINON SI a est inférieur à b
 *  ALORS
 *      ECRIRE a, " est inférieur à ", b
 *  SINON
 *      ECRIRE a, " et ", b, " sont égaux"
 *  FIN SI
 */
if(a > b) {
    System.out.println(a + "est supérieur à " + b);
}
else if(a < b) {
    System.out.println(a + "est inférieur à " + b);
}
else {
    System.out.println(a + " et " + b + " sont égaux");
}
```

Le programme exécutera uniquement le bloc de code situé sous la condition qui renvoie "vrai" (true). Si aucune des conditions ne renvoie "vrai", le bloc de code situé sous l'instruction else est exécuté.

Dans la capture précédente, un seul des 3 affichages sera déclenché. Cela dépendra de la valeur des variables a et b.

SWITCH / CASE

L'instruction switch évalue une expression et, selon le résultat obtenu et le cas associé, exécute les instructions correspondantes.

```
int a = 1;

switch(a)
{
    case 1:
        System.out.println("Gagné, A est égal à 1");
        break;
    case 2:
        System.out.println("Gagné, A est égal à 2");
        break;
    default:
        System.out.println("Perdu, A ne devrait pas être égal à " + a);
        break;
}
```

La valeur de la variable **a** est évaluée.

Si **a** vaut 1, le code sous le cas 1 (case 1) est exécuté.

L'instruction **break** qui suit permet de de sortir du bloc switch courant après exécution du cas 1.

C'est-à-dire que si le cas 1 est exécuté, les autres cas seront ignorés.

Si **a** vaut 2, le code sous le cas 1 est ignoré et le code du cas 2 est exécuté.

Si la valeur de **a** ne correspondant à aucun cas, le cas par défaut (default) est exécuté.

/! Pour les structures conditionnelles, prévoyez **toujours** un cas par défaut :

- **else** pour les instructions if/elseif/else.
- **default** pour les instructions switch/case.

LES STRUCTURES REPETITIVES

Une structure répétitive (boucle) permet d'exécuter un bloc d'instructions plusieurs fois.

Il existe 3 boucles principales qui fonctionnent exactement de la même manière dans à peu près tous les langages de programmation :

- while : TANT QUE
- do … while : FAIRE … TANT QUE
- for : POUR

Une 4ème boucle existe dans de nombreux langages, il s'agit de la boucle foreach (POUR CHAQUE). Cependant, sa syntaxe est très différente selon le langage utilisé.

WHILE

```
while (boolean) {
  ...// Code à exécuter dans la boucle
}
```

Le code est exécuté tant que le booléen est vrai. Si avant l'instruction **while**, le booléen est faux, alors le code de la boucle ne sera jamais exécuté.
La boucle **while** est utilisée lorsque l'on ne connait pas à l'avance le nombre exact de fois que la boucle devra s'exécuter.

```
int a = 1;

while(a < 10) {
    System.out.println(a);
    a++; // incrémentation de a
}
```

/!\ **Vous devez vous assurez que votre programme sortira de la boucle à un moment donné**. Si tel n'est pas le cas, le programme restera "coincé" dans cette **boucle infinie** et finira par planter.

DO … WHILE

```
do {
 ...//Code à exécuter dans la boucle
} while (boolean);
```

La boucle **do...while** est une variante de la boucle **while**. Sa particularité réside dans le fait que la condition est testée après la première exécution de la boucle. Le code est exécuté tant que la condition est satisfaite et est exécuté au moins une fois.

```
int a = 10;

do {
    System.out.println(a);
    a++; // incrémentation de a
}
while(a < 10);
```

/!\ Tout comme pour la boucle while, **Vous devez vous assurez que votre programme sortira de la boucle à un moment donné**. Si tel n'est pas le cas, le programme restera "coincé" dans cette **boucle infinie** et finira par planter.

FOR

```
for (initialisation; condition; modification) {
 ...//Code à exécuter dans la boucle
}
```

La boucle for teste une condition et exécute le bloc de code rattaché à la boucle tant que la condition est remplie.
La boucle for est le plus utilisée lorsque l'on sait combien de fois on souhaite exécuter le bloc de code rattaché à la boucle.

```
int i;

for(i = 0; i < 10; i++) {
    System.out.println(i);
}
```

FONCTIONNEMENT DU FOR

Une boucle for possède 3 paramètres :

- un point de départ
- une condition
- changer le départ

```
FOR ( point de départ ; Condition ; Changer le départ ){

        CODE.....

}
```

Dans le code précédent,

```
int i;

for(i = 0; i < 10; i++) {
    System.out.println(i);
}
```

1. Le point de départ est la valeur de la variable **i**, soit **0**.
2. La valeur de **i** est inférieure à **10**, on exécute donc le bloc de code rattaché au for
3. puis **i** est incrémenté de **1**, et on recommence.
4. **i** vaut désormais **1** et la boucle continue tant que la condition "**i** est inférieure à **10**" est remplie.

On pourrait remplacer ce for par une boucle while :

i est un entier
i ← 0

Tant que **i** est inférieur à 10

  Afficher **i**
  Incrémenter **i**

Fin Tant que

Soit en Java :

```
int i;

for(i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```
int i = 0;

while(i < 10) {
    System.out.println(i);
    i++;
}
```

Ces 2 boucles effectuent le même traitement

LES TABLEAUX

Un tableau est une collection de valeurs. Là où une variable stocke 1 valeur, une collection permet d'en stocker plusieurs.

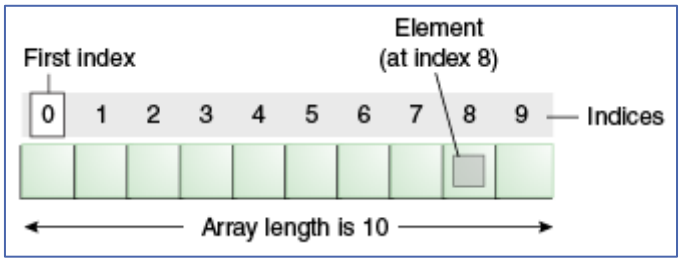
Un tableau stocke toujours des valeurs de même type.
On parlera alors de "tableau de chaines de caractères", "tableau d'entiers ", etc…

Pour déclarer un tableau, on indique le type du tableau suivi de 2 crochets [].

La taille d'un tableau est fixe et définie à sa déclaration. Elle n'est ensuite plus modifiable (sauf en recréant le tableau).

La taille d'un tableau détermine le nombre d'éléments qu'il contient.

Pour lire ou écrire dans une "case" du tableau, on utilise son indice qui représente le numéro de la case du tableau où se trouve une valeur. La 1ère case porte le n°0, la 2ème le n°1 etc… jusqu'à la longueur du tableau. Ainsi, un tableau de 10 éléments contiendra des cases numérotées de 0 à 9 (0 étant la 1ère et 9 la dernière).



Exemple en Java :

```
// Déclaration d'un tableau de chaines de caractères
// Le tableau contient 3 élements (vides par défaut).

String[] tableau = new String[3];

tableau[0] = "Bleu"; // 1er élement du tableau
tableau[1] = "Vert"; // 2ème élement du tableau
tableau[2] = "Jaune"; // 3ème élement du tableau

System.out.println(tableau[1]); // affiche le 2ème élément soit "Vert"
```

Il est également possible déclarer les éléments du tableau en même temps que le tableau lui-même.
Sa longueur sera alors définie selon le nombre d'éléments déclarés.

```
// Déclaration d'un tableau de chaines de caractères
// Le tableau contient 3 élements déclarés à l'initialisation.

String[] tableau = new String[] { "Bleu", "Vert", "Jaune" };

System.out.println(tableau[1]); // affiche le 2ème élément soit "Vert"
```

Vous connaissez désormais la syntaxe de base du langage Java. Entrainez-vous avec quelques exercices d'algorithmes 😊

--- FIN DU DOCUMENT ---