

Les Éléments du Langage

Les types primitifs

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
Integer	<code>byte</code>	8	-128	127	From +127 to -128	<code>byte b = 65;</code>
	<code>char</code>	16	0	$2^{16}-1$	All Unicode characters	<code>char c = 'A';</code> <code>char c = 65;</code>
	<code>short</code>	16	-2^{15}	$2^{15}-1$	From +32,767 to -32,768	<code>short s = 65;</code>
	<code>int</code>	32	-2^{31}	$2^{31}-1$	From +2,147,483,647 to -2,147,483,648	<code>int i = 65;</code>
	<code>long</code>	64	-2^{63}	$2^{63}-1$	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	<code>long l = 65L;</code>
Floating-point	<code>float</code>	32	2^{-149}	$(2-2^{-23}) \cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	<code>float f = 65f;</code>
	<code>double</code>	64	2^{-1074}	$(2-2^{-52}) \cdot 2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	<code>double d = 65.55;</code>
Other	<code>boolean</code>	1	--	--	false, true	<code>boolean b = true;</code>
	<code>void</code>	--	--	--	--	--

Ref. https://en.wikibooks.org/wiki/Java_Programming/Primitive_Types

Les variables (1)

- En Java, les variables sont typées
- Les variables peuvent être déclarées n'importe où dans un bloc.

```
...  
{  
  int x;  
  ...  
  {  
    int y;  
    ...  
  }  
  ...  
}
```

} porté de y

} porté de x

Une seconde variable x dans le second bloc ne peut pas être déclarée.

Affectation

- Opérateurs d'affectation :

=

+=, -=, *=, /=, %=

&=, |=, ^=

Les expressions

- Les expressions respectent l'ordre de priorité du C

.	++	*	+	<<	<	==	&	^	&&		=
[]	--	/	-	>>	>	!=					
()	!	%		>>>	<=						
	~				>=						

- Possibilité d'utiliser une expression **ternaire**

```
int x = 100 ;  
int y = 20 ;  
int maximum = (x > y) ? x : y ;
```

Type casting

- Les affectations non implicites doivent être **castées** (sinon erreur à la compilation).

```
int i = 258;
```

```
long l = i; // ok
```

```
byte b = i; // error: Explicit cast needed to convert int to byte
```

```
byte b = 258; // error: Explicit cast needed to convert int to byte
```

```
byte b = (byte)i; // ok mais b = 2
```

- NB : levé d'ambiguïté entre **float** et **double**
 - **float f = 2564.5 // ERREUR à la compilation (float ou double ?)**
 - **float f = 2564.5f // OK**

Les méthodes

```
TypeRetour nomMethode(listeParamètres) {  
    ...corps...  
}
```

- Le type de retour est un type primitif, une classe ou **void**
- La liste des paramètres peut être vide
- Si le type de retour n'est pas void, la fonction doit se terminer par un **return** ...

Passage de paramètres

- Les paramètres de **type simple** (int, float...) sont passés par **valeur**.
- Les paramètres de type **Objet** ou tableau sont passés par **référence**.

Structures de contrôle (1)

- IF

if(condition) {...} *else* {...}

if(condition) instruction;

if(condition) instruction; *else* instruction;

if(condition) instruction; *else* {...}

if(condition) {...} *else* instruction;

Structures de contrôle (2)

- WHILE

while (condition) instruction;

while (condition) { ... }

Tant que la condition est vraie,
exécuter les instructions du bloc

Structures de contrôle (3)

- DO...WHILE

do {...}*while* (condition);

do instruction *while* (condition);

Exécuter les instructions du bloc **tant que** la condition est vraie

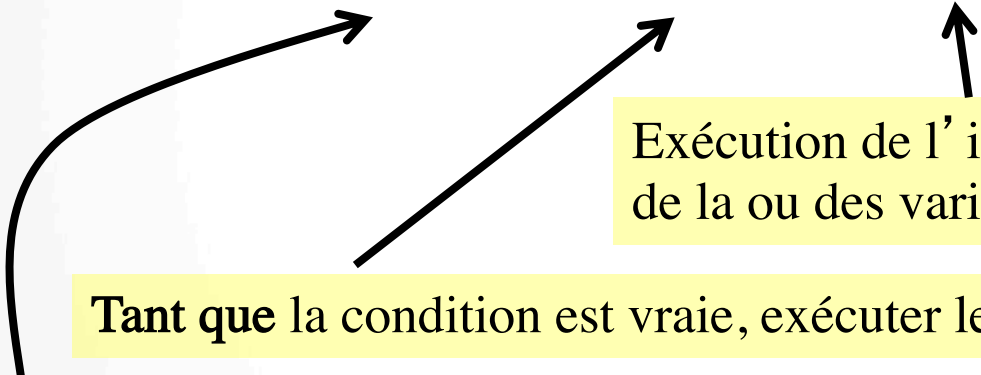
**Les instructions du bloc sont exécutées au moins une fois.
La condition est évaluée après cette première exécution.**

Structures de contrôle (4)

- FOR

for (initialisation; condition; incrémentation) {...}

for (initialisation; condition; incrémentation) instruction;



Exécution de l' instruction d' incrémentation
de la ou des variables de boucle

Tant que la condition est vraie, exécuter les instructions du bloc

Initialisation de la ou des variables de boucle

L' opérateur , peut être utilisé pour les parties initialisation et incrémentation dans le cas où elles nécessiteraient plusieurs instructions.

Ex. *for*(*i*=1, *j*=5; *condition* ; *i*++, *j* += 2) ...

Structures de contrôle (5)

- SWITCH

```
switch (expression) {  
    case constante: instructions; break;  
    case constante: instructions; break;  
    ...  
    default: instructions  
}
```

Structures de contrôle (6)

- BREAK et CONTINUE
 - *break* stoppe une boucle (par exemple, une boucle infinie : la condition d'arrêt est définie à l'intérieure de la boucle et si la condition se réalise alors l'instruction *break* stoppe la boucle et continue en séquence)
 - *continue* passe automatiquement à l'itération suivante sans exécuter les instructions suivantes de la boucle

Entrée-sorties (1)

- Les entrée-sorties sont possibles grâce à deux flux :
 - System.out : flux de sortie (console)
 - System.in : flux d'entrée (clavier)
- `print` et `println` permettent d'afficher à l'écran sans ou avec un retour chariot
- Ex.

```
System.out.println("Hello World");  
  
String name = "Greg" ;  
System.out.println("Hello " + name);
```

Entrée-sorties (2)

- Les caractères de contrôle d'affichage sont les mêmes qu'en C :
 - `\n` : retour chariot + retour à la ligne
 - `\t` : tabulation
 - `\\` : `\`
 - `\"` : `»`
- La saisie au clavier est facilitée grâce à la classe Scanner
 - (cf. TDs)

Code minimal

- Comme en C, une méthode spéciale constitue le point de départ du programme.
- La différence réside dans le fait que tout est objet, on ne peut donc pas définir une méthode globale *main*.

```
public class UneClasse {  
    public static void main(String[] args) {  
    }  
}
```

Exemple de programme

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

Fichier: Hello.java

Compilation: **javac Hello.java** → Hello.class

Exécution: **java Hello**

Les commentaires

- 3 types de commentaires :
 - de type C : `/* commentaire */`
 - de type C++ (commentaire de fin de ligne) :
`// commentaire`
 - pour la documentation (**javadoc**) :
`/** commentaire */`

Les commentaires

- Les commentaires destinés à la documentation du code peuvent être exprimés soit en code HTML soit en utilisant des balises de documentation spécifiques à Java (@...).
- Il existe trois types de commentaires associés aux trois éléments de base : les classes, les attributs des classes et les méthodes des classes.
 - Le commentaire est placé juste avant l'un de ces trois éléments.
- Les commentaires permettent de générer la documentation sous forme de fichiers HTML grâce à la commande *javadoc*.

Les commentaires

```
/** Exemple de classe simple
 * @author Greg
 * @version 1.0
 */
public class Hello {

    /** Methode principale qui affiche un message sur la sortie standard
     * @param args : arguments de la ligne de commande, non utilisés ici
     */
    public static void main(String[] args) {
        System.out.println("Hello World !!!"); // Dit bonjour
    }
}
```

Les tableaux

- Les tableaux sont la plus simple des solutions pour stocker et accéder à entités de même type.
- Un tableau est une structure linéaire de taille **fixe**.
- La taille ne peut pas changer durant la vie du tableau.
- Un tableau contient soit des objets, soit des valeurs d'un même type primitif.

Les tableaux

- La déclaration se fait à l'aide des crochets [] :
`type[] tab1, tab2 ;`
`type tab3[] ;`
NB : `int tab1[], entier1 ;` // crée un tableau et un entier
- Il faut maintenant initialiser le tableau = fixer la taille et éventuellement stocker les valeurs.
 - Exemple : tableau de 7 entiers :
`int v[]={1, 2, 3, 4, 5, 6, 7};`
- L'indice du tableau est compris entre 0 et `v.length-1`.

Les tableaux

```
public class TestTab {  
    public static void main(String[] args) {  
        int[] t1 = { 1, 2, 3, 4, 5 } ;  
        int[] t2 ;  
  
        t2 = t1 ; // t1 et t2 referencent maintenant le même tableau  
  
        for (int i = 0; i < t2.length ; i++) t2[i]++ ;  
  
        for (int i = 0; i < t1.length; i++) {  
            System.out.println("t1[" + i + "] = " + t1[i]);  
        }  
    }  
}
```

Les tableaux

- Il est possible de créer dynamiquement un tableau grâce à l'instruction **new**.
- La création dynamique est utilisée lorsque la taille du tableau n'est pas connue lors de l'écriture du programme ou lorsqu'elle dépend du contenu d'une ou plusieurs variables.

Les tableaux

```
import java.util.Random ; // Pour utiliser le generateur aleatoire

public class TestTab {

    public static void main(String[] args) {
        int tab[] ; // Declaration de tableau d'entiers
        Random rand = new Random() ; // Generateur aleatoire
        final int MAX = 20 ; // Definition de constante

        int nb_aleatoire = rand.nextInt(MAX) ; // Tirage d'un nombre
        tab = new int[nb_aleatoire] ; // Instanciation du tableau

        System.out.println("Taille du tableau : " + tab.length);
        for (int i=0 ; i<tab.length ; i++){ // Parcours du tableau
            System.out.println("Contenu indice : " + i + " > " + tab[i]);
        }
    }
}
```

Les tableaux multidimensionnels

- En Java, un tableau multidimensionnel n'est qu'un tableau dont les éléments sont eux-mêmes des tableaux.
- Ceci implique, par exemple, que pour connaître la taille du tableau selon la deuxième dimension, il faut écrire :
`t[i].length`

Les tableaux multidimensionnels

Ex. 1 :

```
int tab2D[][] = {  
    { 11, 12, 13 },  
    { 21, 22, 23 }  
};  
for (int i = 0; i < tab2D.length; i++) {  
    for (int j = 0; j < tab2D[i].length; j++) {  
        System.out.println("Case [" + i + ", " + j + "] > " + tab2D[i][j]);  
    }  
}
```

Ex. 2 :

```
Random rand = new Random(); // Generateur aleatoire  
int tab2D[][] = new int[3][5]; // Instanciation d'un tableau 3 x 5 ;  
for (int i = 0; i < tab2D.length; i++) {  
    for (int j = 0; j < tab2D[i].length; j++) {  
        tab2D[i][j] = rand.nextInt(); // Remplissage avec un nombre aleatoire  
    }  
}
```