

Java : Syntaxe de Base

1.	Les constantes et les variables.....	2
1.1.	Les constantes	2
1.2.	Les variables	2
2.	Les types primitifs	3
2.1.	Les entiers	3
2.2.	Les réels	3
2.3.	Les caractères	4
2.4.	Les booléens.....	4
2.5.	Les règles de priorité.....	5
2.6.	Les conversions de type (ou transtypage).....	5
3.	Les instructions élémentaires	7
3.1.	Les affectations	7
3.2.	Les sorties	7
3.3.	Les entrées.....	8
3.4.	Les sélections	8
3.5.	Les itérations	10
4.	Les tableaux.....	12
5.	Les chaînes de caractères	15
6.	Les méthodes	17

1. Les constantes et les variables

1.1. Les constantes

Une constante se déclare en indiquant son type et en utilisant le mot-clé **final**.

Si plusieurs constantes sont de même type, elles peuvent être déclarées ensemble.

Les constantes peuvent être ou non initialisées lors de leur déclaration, mais leur valeur n'est pas modifiable.

Les constantes peuvent être déclarées n'importe où dans un programme. Elles sont visibles dans le bloc d'instructions dans lequel elles sont déclarées.

```
final int NB_MAX = 20;
```

```
final int NB_MOY, NB_MIN = 6;
```

....

```
NB_MOY = 8;
```

```
NB_MAX = 30; //KO
```

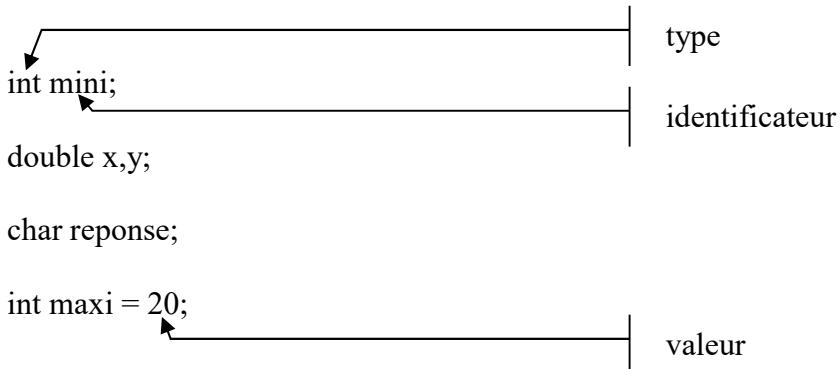
1.2. Les variables

Une variable se déclare en indiquant le type puis l'identificateur de la variable.

Si plusieurs variables sont de même type, elles peuvent être déclarées ensemble.

Les variables peuvent être initialisées lors de leur déclaration.

Les variables peuvent être déclarées n'importe où dans un programme. Elles sont visibles dans le bloc d'instructions dans lequel elles sont déclarées.



2. Les types primitifs

2.1. Les entiers

Il existe quatre types entiers :

	taille	valeur minimale	valeur maximale
byte	8 bits i.e. 1 octet	-128 (Byte.MIN_VALUE)	127 (Byte.MAX_VALUE)
short	16 bits i.e. 2 octets	-32 768 (Short.MIN_VALUE)	32 767 (Short.MAX_VALUE)
int	32 bits i.e. 4 octets	-2 147 483 648 (Integer.MIN_VALUE)	2 147 483 647 (Integer.MAX_VALUE)
long	64 bits i.e. 8 octets	-9 223 372 036 854 775 808 (Long.MIN_VALUE)	9 223 372 036 854 775 807 (Long.MAX_VALUE)

Les opérateurs arithmétiques applicables sur les entiers sont :

- les opérateurs unaires : identité, opposé, post et pré-incrémantation, post et pré-décrémantation,
- les opérateurs binaires : addition, soustraction, multiplication, division entière et reste de la division entière.

Les opérateurs relationnels applicables sur les entiers sont l’infériorité stricte, l’infériorité, la supériorité stricte, la supériorité, l’égalité et la différence.

Opérateurs arithmétiques :

+3	a++	a--	3 + 5	5 * 2	5 / 2
-45	++a	--a	3 - 5		5 % 2

Opérateurs relationnels :

0 < 1	1 > 0	0 == 0
0 <= 1	1 >= 0	1 != 0

2.2. Les réels

Il existe deux types réels :

	taille	valeur absolue minimale	valeur absolue maximale
float	32 bits i.e. 4 octets (précision : 7 chiffres significatifs)	1.4*10 ⁻⁴⁵ (Float.MIN_VALUE)	3.4*10 ³⁸ (Float.MAX_VALUE)
double	64 bits i.e. 8 octets (précision : 15 chiffres significatifs)	4.9*10 ⁻³²⁴ (Double.MIN_VALUE)	1.8*10 ³⁰⁸ (Double.MAX_VALUE)

Les opérateurs arithmétiques applicables sur les réels sont l’identité, l’opposé, l’addition, la soustraction, la multiplication et la division. Les opérateurs relationnels applicables sur les réels sont les mêmes que ceux définis sur les types entiers.

Opérateurs arithmétiques :

+45.5	+3.1e12	+3.1E12	35.5E-1 + 45.5 => 49.05
-45.5			45.5 - 3.5

Opérateurs relationnels :

cf. types entiers

2.3. Les caractères

Le jeu de caractères utilisé pour coder les caractères est **Unicode**.

Un caractère peut se définir soit en utilisant des apostrophes, soit en utilisant son code hexadécimal dans le jeu Unicode (ex : ‘\u0041’ correspond au caractère ‘A’).

Les opérateurs relationnels applicables aux types entiers sont également applicables aux caractères.

char	16 bits i.e. 2 octets (65 536 valeurs)
-------------	--

‘a’	‘ç’	‘A’	‘0’	‘\u0041’
\n	passage à la ligne		\t	tabulation
\\"	antislash		\'	apostrophe
			\"	guillemets

Opérateurs relationnels :

cf. types entiers

2.4. Les booléens

Nom du type : **boolean**
 Valeurs possibles : true false

Les opérateurs logiques sont la négation, la disjonction, la disjonction exclusive et la conjonction. On peut également utiliser la disjonction et la conjonction conditionnelles.

Opérateurs booléens

p	q	!p	p q	p ^ q	p & q
true	true	false	true	false	true
true	false		true	true	false
false	true	true	true	true	false
false	false		false	false	false

Opérateurs conditionnels

- p || q si p est vrai, q n'est pas évalué
- p && q si p est faux, q n'est pas évalué

2.5. Les règles de priorité

Dans le tableau ci-dessous on a classé les opérateurs selon leur ordre de priorité : des opérateurs les moins prioritaires en bas aux opérateurs les plus prioritaires en haut ; les opérateurs de même priorité sont rangés sur la même ligne du tableau.

+	() !, --, ++, - et + unaires * / % + - < <= > >= == != & ^ && = += -= *= /= %=-
---	--

Exemples d'évaluation :

- $3 + 5 / 2$ équivaut à $3 + (5 / 2)$ => 5
- $(3+5) / 2$ => 4
- $3 + 5.0 / 2$ => 5.5
- $++a == 0 \wedge a <= b \ \&\& c != 0$
équivaut à
 $((++a) == 0) \wedge (a <= b) \ \&\& (c != 0)$

2.6. Les conversions de type (ou transtypage)

On peut convertir n'importe quel type élémentaire en n'importe quel autre type élémentaire, sauf les booléens. Certaines conversions sont implicites : elles sont effectuées automatiquement par Java. D'autres, les conversions explicites, doivent être explicitement demandées dans le programme.

• conversions implicites

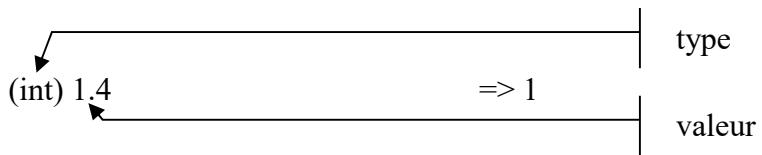
La conversion d'une donnée vers un type dont la représentation est plus grande se fait implicitement, sans perte d'informations.

règle : byte → short → int → long → float → double

3 + 6.0 / 2 => conversion de 2 et 3 en réels
 int x = 'A' => la valeur de x est 65

• conversions explicites (« cast »)

La conversion d'une donnée vers un type dont la représentation est plus petite doit être explicitement signifiée. Elle se note en indiquant entre parenthèses le type dans lequel on veut convertir la donnée.



int x;
x = (int) 1.4; \Rightarrow la valeur de x est 1

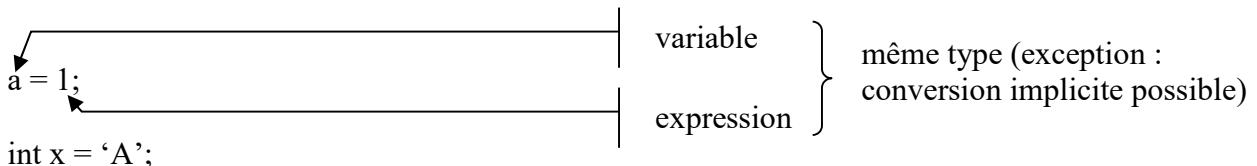
char c;
c = 65 ; \Rightarrow la valeur de c est 'A'
c = (char) 65.5; \Rightarrow la valeur de c est 'A'

3. Les instructions élémentaires

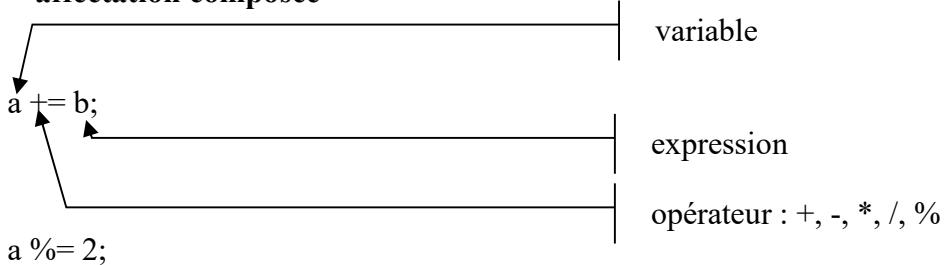
3.1. Les affectations

Pour assigner une valeur à une variable ou une constante, on peut utiliser soit l'opérateur d'affectation simple, soit l'un des opérateurs d'affectation composée.

- **affectation simple**



- **affectation composée**



3.2. Les sorties

En mode console, pour afficher un message, on peut utiliser les méthodes `print` ou `println` en les appliquant à la sortie standard du système. Ces méthodes ne prennent qu'un seul paramètre (chaîne de caractères, nombre, etc.).

La différence entre les deux méthodes est que la méthode `println` permet d'ajouter un passage à la ligne à la fin du message.

Pour afficher, en une seule instruction, plusieurs chaînes de caractères on peut utiliser l'opérateur de concaténation (cf. **5. Les chaînes de caractères**)

```
int n=3;
System.out.print (n);
```

A bracket under `System.out.print (n);` points to a vertical line labeled "sortie standard du système".

```
System.out.print ("Le carré de " + n + " vaut : ");
System.out.println (n*n);
```

```
System.out.println ("Le carré de " + n + " vaut : ");
System.out.println (n*n);
```

```
System.out.println ("Le carré de " + n + " vaut : n*n ");
```

3.3. Les entrées

En mode console, pour lire des valeurs, on peut utiliser les méthodes de la classe Scanner.
 Il faut : déclarer une variable de type Scanner (nommée sc dans l'exemple ci-dessous), la lier à l'entrée standard du système (en utilisant l'instruction new), puis appliquer à la variable la méthode correspondant au type de la valeur que l'on souhaite lire (nextLine, nextInt, etc.).

```

Scanner sc;
String ch;
int x;
float y;
boolean b;

sc = new Scanner (System.in);                                | entrée standard du système

System.out.println("Entrer votre prénom");
ch = sc.nextLine();

System.out.println("Entrer dans l'ordre la valeur de x puis celle de y puis celle de b : ");
x = sc.nextInt();                                         //existent aussi : nextByte, nextShort, nextLong
y = sc.nextFloat();                                       //existent aussi : nextDouble
b = sc.nextBoolean();
  
```

3.4. Les sélections

Pour soumettre un traitement (i.e. une suite d'instructions) à la valeur d'une condition, il existe trois sortes de sélection.

- **La sélection simple**

La condition d'une sélection se note entre parenthèses, le traitement effectué selon la valeur de la condition se note entre accolades (ces accolades sont optionnelles si le traitement ne contient qu'une seule instruction).

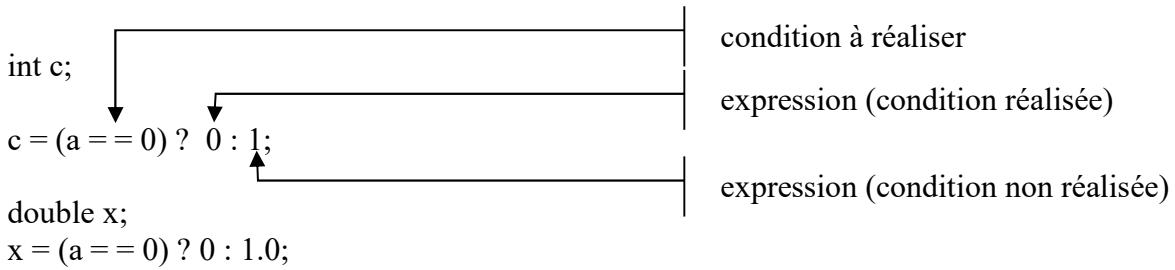
```

if (a < 0) {                                              | condition à réaliser
    if (b != 0)      c = -1;
    else            c = 0;
}
else //a>0
    if (b == 0)    c = 0;
    else
        if (b>0)   c = 1;
        else          c = -1;                                } suite d'instructions facultatif
  
```

- **L'expression conditionnelle**

Pour écrire une sélection, il existe également un opérateur ternaire.

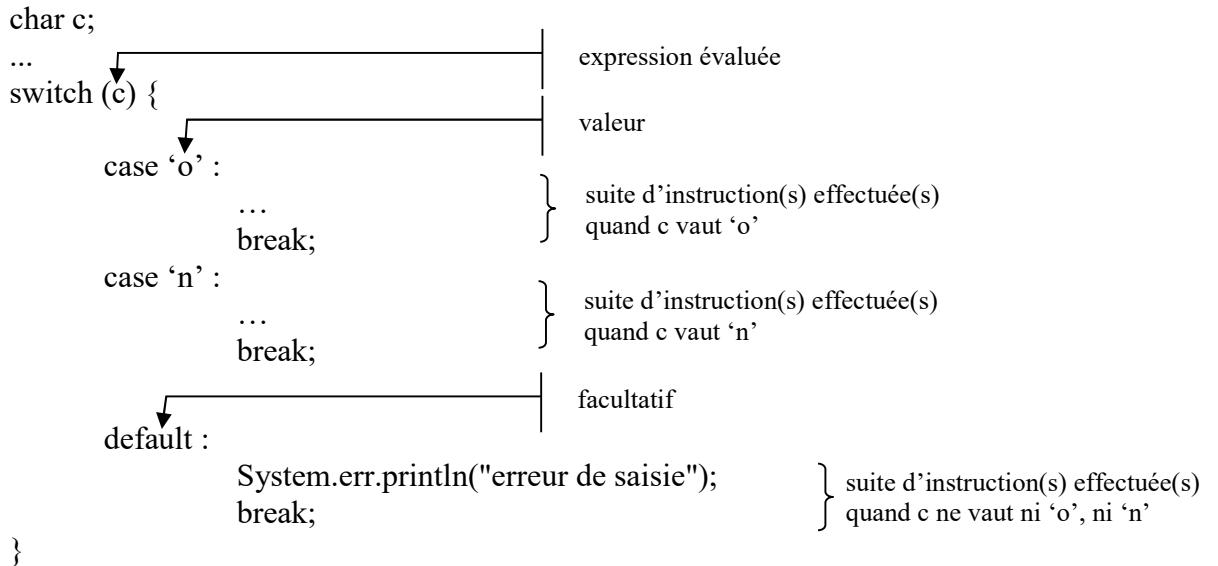
La condition se note entre parenthèses, suivie d'un point d'interrogation. Les instructions à effectuer lorsque la condition est vraie et lorsqu'elle est fausse sont notées de part et d'autre de deux-points.
 Cet opérateur renvoie le résultat de l'évaluation des instructions exécutées.



- **Le choix multiple**

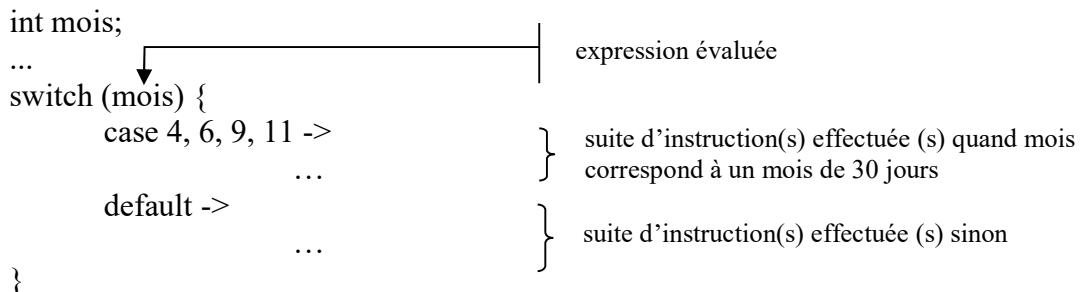
Le choix multiple est mis en œuvre grâce à l'opérateur *switch*. Il peut être utilisé avec des entiers, des caractères ou des chaînes de caractères.

Dans le premier exemple ci-dessous, on énumère les différentes valeurs possibles de la variable *c* et, pour chaque cas, on écrit le traitement qui doit être exécuté. L'instruction *break* permet d'indiquer l'arrêt du traitement.



Remarque : si le traitement associé à un *case* ne contient pas de *break*, le traitement associé au *case* suivant sera exécuté également.

Dans le second exemple, pour indiquer le traitement associé à chaque *case*, on utilise la flèche (*->*) à la place des deux-points. Cela permet d'éviter d'avoir à utiliser l'instruction *break*.



On note qu'il est possible de traiter, dans un même *case*, plusieurs valeurs en même temps.

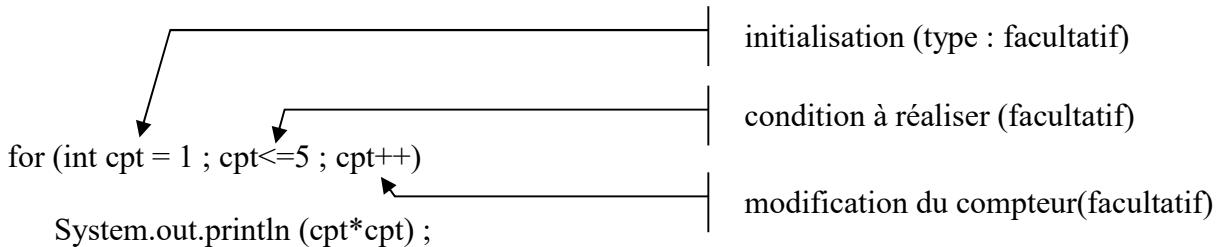
3.5. Les itérations

Pour répéter un traitement (i.e. une suite d'instructions) plusieurs fois, suivant la valeur d'une condition, il existe trois sortes d'itération.

- **La boucle « Pour »**

La condition et la modification du compteur sont optionnelles.

Le compteur peut être déclaré dans la partie initialisation, il n'est alors visible que dans le corps de la boucle.



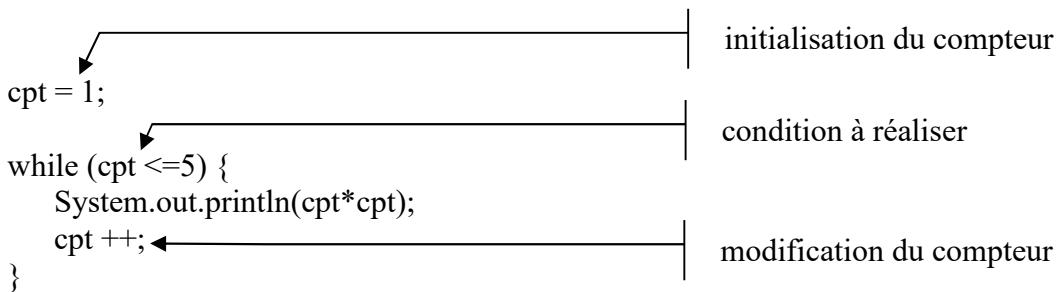
```

int cpt2, cpt3;
for (cpt2 =5 ; cpt2 > 0 ; cpt2--)
    System.out.println (cpt2*cpt2);
  
```

```

for (cpt3 =5 ;;) {
    System.out.println (cpt3*cpt3);
    cpt3--;
    if (cpt3 == 0) break;
}
  
```

- **La boucle « Tant que ... faire »**



Remarque : si la condition est initialement fausse, les instructions contenues dans le corps de la boucle ne sont jamais exécutées.

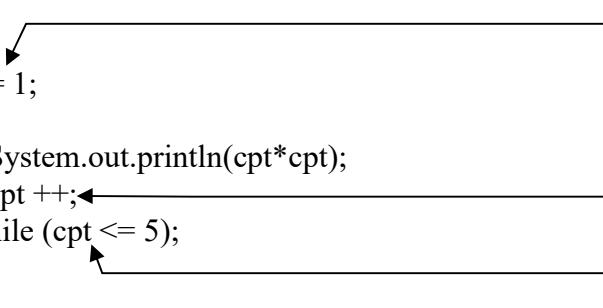
- **La boucle « Faire ... tant que ...»**

Les instructions contenues dans le corps de la boucle sont toujours exécutées au moins une fois, quelle que soit la valeur initiale de la condition.

```

cpt = 1;
do {
    System.out.println(cpt*cpt);
    cpt++;
} while (cpt <= 5);

```



initialisation du compteur

modification du compteur

condition à réaliser

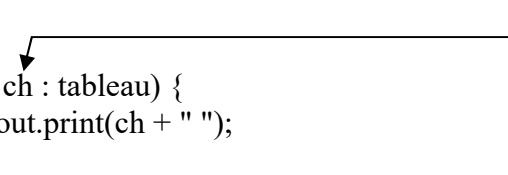
- **La boucle « Pour chaque »**

Cette boucle permet de parcourir les différents éléments d'un ensemble (un tableau, une liste, etc.). Dans l'exemple ci-dessous, on affiche le contenu d'un tableau de chaîne de caractères.

```

String tableau[] = {"un", "deux", "trois"};
System.out.println("Affichage du contenu du tableau");
for (String ch : tableau) {
    System.out.print(ch + " ");
}

```



ch correspond successivement à chacun des éléments du tableau

4. Les tableaux

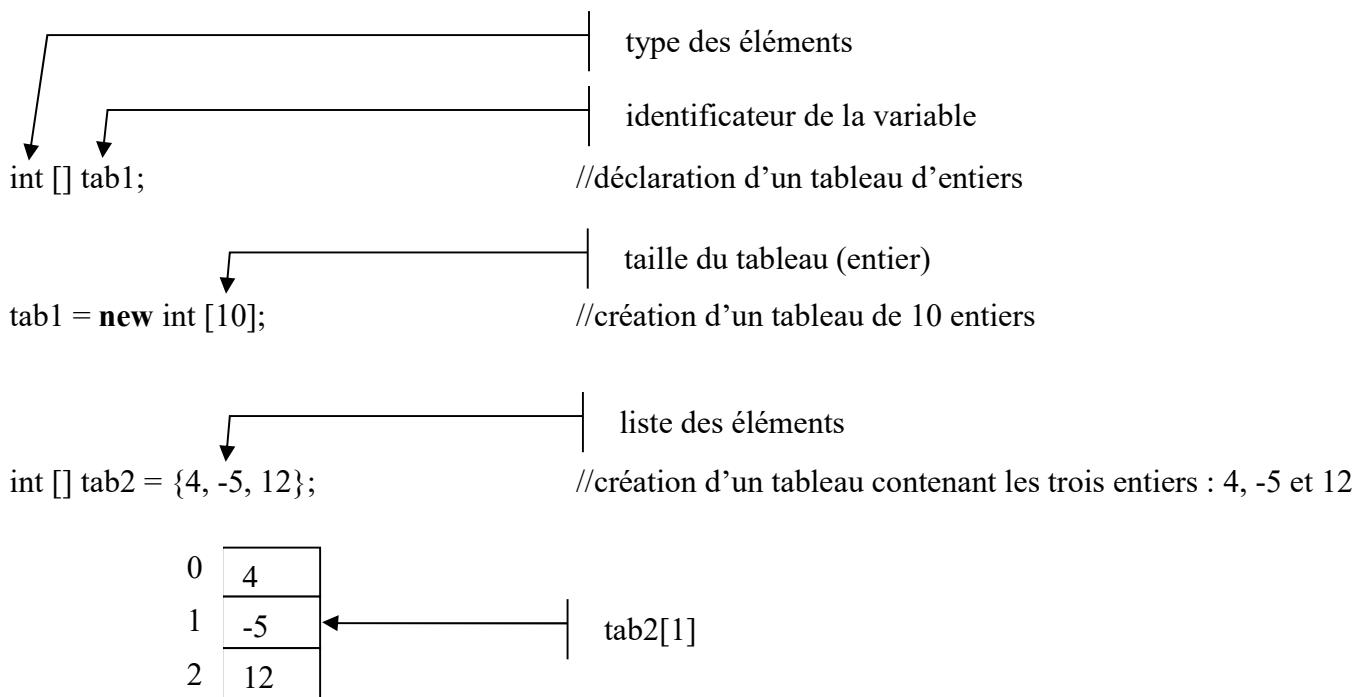
Un tableau est un ensemble indicé d'éléments qui sont tous de même type.

Il se déclare en indiquant le type des éléments et en utilisant des crochets. Il faut ensuite créer le tableau, soit en utilisant l'instruction new et en indiquant le nombre d'éléments du tableau, soit en initialisant directement le tableau.

Dans l'exemple ci-dessous, les variables tab1 et tab2 correspondent à des tableaux unidimensionnels d'entiers. Le tableau tab1 est créé en utilisant l'instruction new et en indiquant sa taille, le tableau tab2 est créé en indiquant directement les entiers qu'il contient.

Une fois le tableau créé, sa taille n'est plus modifiable. De plus, la création du tableau est obligatoire avant toute manipulation.

Pour accéder à un élément du tableau, on utilise des crochets en indiquant l'indice de l'élément. Le premier élément a comme indice 0, le i ème élément du tableau a donc comme indice i-1.



Possible également :

```

int [] tab2 ;
...
tab2 = new int [ ]{4, -5, 12};
  
```

Un tableau peut aussi être multidimensionnel. Il correspond alors en fait à un tableau de sous-tableaux, qui peuvent être de même taille ou de taille différente.

Dans l'exemple ci-dessous, tab3 correspond à un tableau de 10*5 entiers, tab4 correspond à un tableau de 2*3 entiers, le tableau tab5 contient un sous-tableau de 4 entiers, un sous-tableau de 2 entiers et un sous-tableau de 3 entiers.

```
int [][] tab3; //tableau de dimension 2
tab3 = new int [10] [5]; //création d'un tableau 10*5
```

	0	1	2	3	4
0					
1					
...
9					

tab3[1][2]

```
int [] [] tab4 = { {1,2,3} , {4,5,6} }; //création d'un tableau 2*3 en indiquant son contenu
```

	0	1	2
0	1	2	3
1	4	5	6

```
int [] [] tab5 = { {1,2,3,4}, {5,6}, {7,8,9} }; //création d'un tableau contenant 3 sous-tableaux de taille  
//différente
```

0	1	2	3	4
1	5	6		
2	7	8	9	

Possible également :

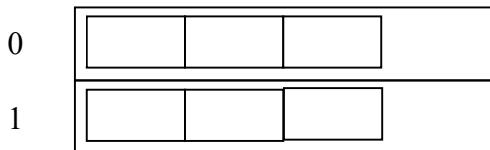
```
int [] [] tab5
tab5 = new int [] [] {{1,2,3,4}, {5,6}, {7,8,9}};
```

```
int [] [] tab6 = new int [3] [];
tab6[0] = new int [4]; //création d'un sous-tableau contenant 4 entiers
tab6[1] = new int [2]; //création d'un sous-tableau contenant 2 entiers
tab6[2] = new int [3]; //création d'un sous-tableau contenant 3 entiers
```

0				
1				
2				

```
int [][] tab7 = new int [2][];
for (int lig=0; lig<tab7.length; lig++)
    tab7[lig] = new int[3];
```

//création d'un tableau contenant 2 sous-tableaux
taille du tableau tab7
//création d'un tableau contenant 3 entiers



• Affectations & Comparaisons

Lorsque l'on déclare une variable de type tableau, la variable correspond en réalité à une référence sur le tableau.

Ainsi si, comme dans l'exemple ci-dessous, on affecte à une variable tab1 une autre variable tab2, c'est la référence contenue dans tab2 qui est recopiée dans la variable tab1. Les variables tab1 et tab2 contiennent donc toutes les deux la même référence, autrement dit elles correspondent au même tableau.

De même, si on compare deux variables de type tableau, ce sont les références contenues dans ces deux variables qui sont comparées et non les éléments des deux tableaux.

Ainsi, dans l'exemple ci-dessous, les trois tableaux tab1, tab2 et tab3 contiennent les mêmes éléments, mais seule la comparaison de tab1 et tab2 retourne la valeur true.

```
int [] tab1 = new int[8];
int [] tab2 = {40,50};
int [] tab3 = {40,50};
```

affectation des références (et non recopie des éléments)

```
tab1 = tab2;
```

comparaison des références (et non comparaison des éléments)

```
System.out.println (tab1 == tab2); // résultat obtenu : true
System.out.println (tab3 == tab2); // résultat obtenu : false
```

5. Les chaînes de caractères

Il existe un type prédéfini pour les chaînes de caractères : la classe String.

Les chaînes se notent entre guillemets.

On peut initialiser une variable de type String soit en lui affectant directement une valeur, soit en utilisant l'instruction new.

On peut utiliser l'opérateur de concaténation pour concaténer deux chaînes de caractères. On peut également utiliser cet opérateur pour concaténer une chaîne de caractères et une valeur d'un type primitif. Dans ce cas, la conversion de cette valeur en chaîne de caractères est implicite.

Pour insérer des guillemets ou des caractères spéciaux dans une chaîne, on peut utiliser le caractère \ ou un code hexadécimal du jeu Unicode.

```

String chaine1;
String chaine2 = "pere";
String chaine3 = new String ("pere");

chaine1 = chaine2 + " et mere d'Alice !" ;
                                //résultat obtenu : pere et mere d'Alice !
String chaine4 = "la mère d'Alice a " + 40 + " ans";
                                //résultat obtenu : la mère d'Alice a 40 ans
String chaine5 = "la 1ère lettre du mot \"pere\" est \n 'p'";
                                //résultat obtenu : la 1ère lettre du mot " pere " est
                                //                            'p'

String message = "Continuez \u0028 o/n \u0029 ?";
                                //résultat obtenu : Continuez (o/n) ?

String message2 = """
    mon texte est trop trop trop long
    pour que je puisse l'écrire
    sur une seule ligne """;
                                //résultat obtenu : mon texte est trop trop trop long
                                //pour que je puisse l'écrire
                                //sur une seule ligne
    
```

Quelques méthodes intéressantes sur les chaînes de caractères :

- length : permet d'obtenir le nombre de caractères de la chaîne.
- charAt : permet d'obtenir un caractère de la chaîne en donnant son indice (les caractères sont numérotés à partir de 0).
- indexOf : permet d'obtenir l'indice de la première occurrence d'un caractère dans la chaîne.
- subString : permet d'obtenir une sous-chaîne de la chaîne en donnant les indices de début et de fin.
- equals : permet de tester si deux chaînes sont identiques.
- compareTo : permet de comparer deux chaînes en utilisant l'ordre lexicographique. La méthode retourne 0 si les deux chaînes sont identiques, un entier négatif si la première chaîne est inférieure (dans l'ordre lexicographique) à la seconde, un entier positif sinon.
- isBlank : permet de tester si une chaîne est vide ou ne contient que des espaces.

chaine2.length()	//résultat obtenu : 4
chaine1.charAt(2)	//résultat obtenu : r
chaine1.indexOf('e')	//résultat obtenu : 1
chaine1.substring(0,6)	//résultat obtenu : pere e
chaine1.equals(chaine2)	//résultat obtenu : false
chaine1.compareTo(chaine2)	//résultat obtenu : 18
chaine1.isBlank()	//résultat obtenu : false

Pour convertir en chaîne de caractères une valeur d'un type primitif, ou inversement, on peut utiliser les méthodes `valueOf` et `toString`.

```

String ch;
ch = String.valueOf(20);           ch = Integer.toString(20);
ch = String.valueOf(23e-45);       ch = Double.toString(23e-45);
ch = String.valueOf(true);         ch = Boolean.toString(true);
ch = String.valueOf('b');          ch = Character.toString('b');

int nb = Integer.valueOf("20");
double x = Double.valueOf("23e-45");
boolean b = Boolean.valueOf("true");
char c = Character.valueOf("b");
  
```

6. Les méthodes

- Déclaration d'une méthode

```

void nom_de_la_methode ( type1 argument1, ... , typen argumentn ) {
    liste_d_instructions
}

type_de_donnée nom_de_la_methode ( type1 argument1, ... , typen argumentn ) {
    liste_d_instructions
    return valeur;
}

```

Lorsque la méthode ne retourne aucune valeur, on fait précéder le nom de la méthode par le mot clé **void**. Lorsqu'elle retourne une valeur, on doit indiquer le type de cette valeur.

Après le nom de la méthode, on écrit la liste des arguments en indiquant, dans cet ordre, leur type et leur nom. La liste des arguments peut être vide.

- Passage des paramètres :

Le mode de passage des paramètres est toujours le passage par valeur (il n'existe pas de passage par adresse). Autrement dit, lorsqu'une méthode est appelée, elle n'a accès qu'à des copies des valeurs des paramètres qui lui sont passés.

Ainsi, lorsqu'une variable de type primitif est passée en paramètre à une méthode, cette méthode ne peut pas modifier la valeur de la variable.

Lorsqu'une variable de type tableau est passée en paramètre à une méthode, la méthode ne peut pas modifier la référence contenue dans la variable, mais elle peut cependant modifier le contenu du tableau (puisque elle a accès à une copie de la référence sur ce tableau).

Il en est de même pour les objets passés en paramètre (cf. chapitre suivant).

```

... void multiplier (int tab[]){
    for (int i=0 ;i<tab.length ;i++){
        tab[i]=tab[i]*2 ;
    }
}

... void main(String[] args){
    int tab1[]={1,2,3} ;
    multiplier(tab1) ;
    for (int i=0 ;i<tab1.length ;i++){
        System.out.print(tab1[i]+ " ") ;           //résultat obtenu : 2 4 6
    }
}

```