

Introduction à la programmation en JAVA



Table des matières

- I. Introduction à Java et historique du langage
- II. Notre outil de développement : *Eclipse Mars*
- **III. Le langage Java et sa syntaxe**
- IV. La POO avec Java
- V. API Java
- VI. La gestion des exceptions
- VII. Les collections
- VIII. La sérialisation

Le langage Java et sa syntaxe



Aperçu du chapitre

- I. Commenter son code source**
- II. Les mots-clés (mots réservés) et les identificateurs en Java**
- III. Types primitifs et types de références**
- IV. Arithmétique et opérateurs**
- V. Expressions, instructions et blocs**
- VI. Instruction de branchement et de contrôle**
- VII. Les tableaux et la classe String**

I . Commenter son code source

Il existe trois façon d'inclure des commentaires dans un code source en Java:

1. `//` devant une ligne de code

Ex: `//System.out.println("Hello World !");`

2. `/*` et `*/` autour du texte

Ex: `/*System.out.println("Hello World !");*/`

3. `/**` et `*/` autour du texte

Ex: `/**@Auteur = moi*/`

Ce type de commentaire est utilisé pour créer des commentaires que l'exécutable javadoc pourra traiter afin de produire une documentation.

L'usage des commentaires est **primordiale**

Aperçu du chapitre

- I. Commenter son code source
- II. Les mots-clés (mots réservés) et les identificateurs en Java**
- III. Types primitifs et types de références
- IV. Arithmétique et opérateurs
- V. Expressions, instructions et blocs
- VI. Instruction de branchement et de contrôle
- VII. Les tableaux et la classe String

II . Les mots-clés (mots réservés) et les identificateurs en Java

Liste des **mots-clés** en Java:

<code>abstract</code>	<code>else</code>	<code>instanceof</code>	<code>strictfp</code>	<code>while</code>	<code>boolean</code>	<code>false</code>
<code>assert (JDK 1.4)</code>	<code>enum (JDK 1.5)</code>	<code>interface</code>	<code>super</code>		<code>byte</code>	<code>null</code>
<code>break</code>	<code>extends</code>	<code>native</code>	<code>switch</code>		<code>char</code>	<code>true</code>
<code>case</code>	<code>final</code>	<code>new</code>	<code>synchronized</code>		<code>double</code>	
<code>catch</code>	<code>finally</code>	<code>package</code>	<code>this</code>		<code>float</code>	
<code>class</code>	<code>for</code>	<code>private</code>	<code>throw</code>		<code>int</code>	
<code>const (Non utilisé)</code>	<code>goto (Non utilisé)</code>	<code>protected</code>	<code>throws</code>		<code>long</code>	
<code>continue</code>	<code>if</code>	<code>public</code>	<code>transient</code>		<code>short</code>	
<code>default</code>	<code>implements</code>	<code>return</code>	<code>try</code>		<code>void</code>	
<code>do</code>	<code>import</code>	<code>static</code>	<code>volatile</code>			

Chacun de ces mots a une fonctionnalité bien précise.

II . Les mots-clés (mots réservés) et les identificateurs en Java

Un **identificateur** permet de désigner :

- une **variable** : élément repéré par son nom (l'identificateur) et qui peut contenir des données. Ces données pourront être modifiées lors de l'exécution du programme.
- une **méthode** : voir l'orienté objet en Java
- une **classe** : voir l'orienté objet en Java
- ...

II . Les mots-clés (mots réservés) et les identificateurs en Java

Un identificateur doit répondre aux **règles** suivantes:

- Avoir une **taille** inférieure ou égale à **247 caractères**
- **Commencer** par une **lettre**, un dollar « \$ » ou un underscore « _ »
- Ne peuvent **pas commencer par un chiffre**
- Caractères spéciaux autres que « \$ » et « _ » interdits
- Ne peuvent **pas** comporter **d'espaces**
- Peuvent contenir des mots-clés mais ne doivent **pas être des mots-clés**
- Sensible à la **casse**

II . Les mots-clés (mots réservés) et les identificateurs en Java

Un identificateur doit aussi répondre à des **conventions de codages** particulières :

- **Classes et interfaces :**

Composés d'un ou plusieurs mots accolés, chaque mot commence par une majuscule

Exemple : LivreDeCompte

- **Variables et méthodes :**

Composés d'un ou plusieurs mots accolés, chaque mot commence par une majuscule sauf le premier qui commence par une lettre minuscule

Exemple : nombreClients

II . Les mots-clés (mots réservés) et les identificateurs en Java

- **Constantes :**

Noms complètement en majuscule avec le caractère « _ » pour séparer les mots

Exemple : TAILLE_MAX

- **Packages :**

Noms complètement en minuscules

Exemple : monpackage

II . Les mots-clés (mots réservés) et les identificateurs en Java - Exercices

Les **identificateurs** ci-dessous sont-ils valables oui ou non?

Variables :

1. int **a**;
2. float **1number**;
3. Object **_myObject**;
4. Test **mon test**;
5. Object **double**;

Méthodes :

1. public void **myMethod()** { ... }
2. private static int **\$gainMoney()** { ... }
3. public static int **gainMoney3()** { ... }

Classes :

1. public class **maCl@ssePerso** { ... }
2. Public class **customClass** { ... }

Aperçu du chapitre

- I. **Commenter son code source**
- II. **Les mots-clés (mots réservés) et les identificateurs en Java**
- III. **Types primitifs et types de références**
- IV. **Arithmétique et opérateurs**
- V. **Expressions, instructions et blocs**
- VI. **Instruction de branchement et de contrôle**
- VII. **Les tableaux et la classe String**

III . Types primitifs et types de références

Le Java est un langage **fortement typé**.

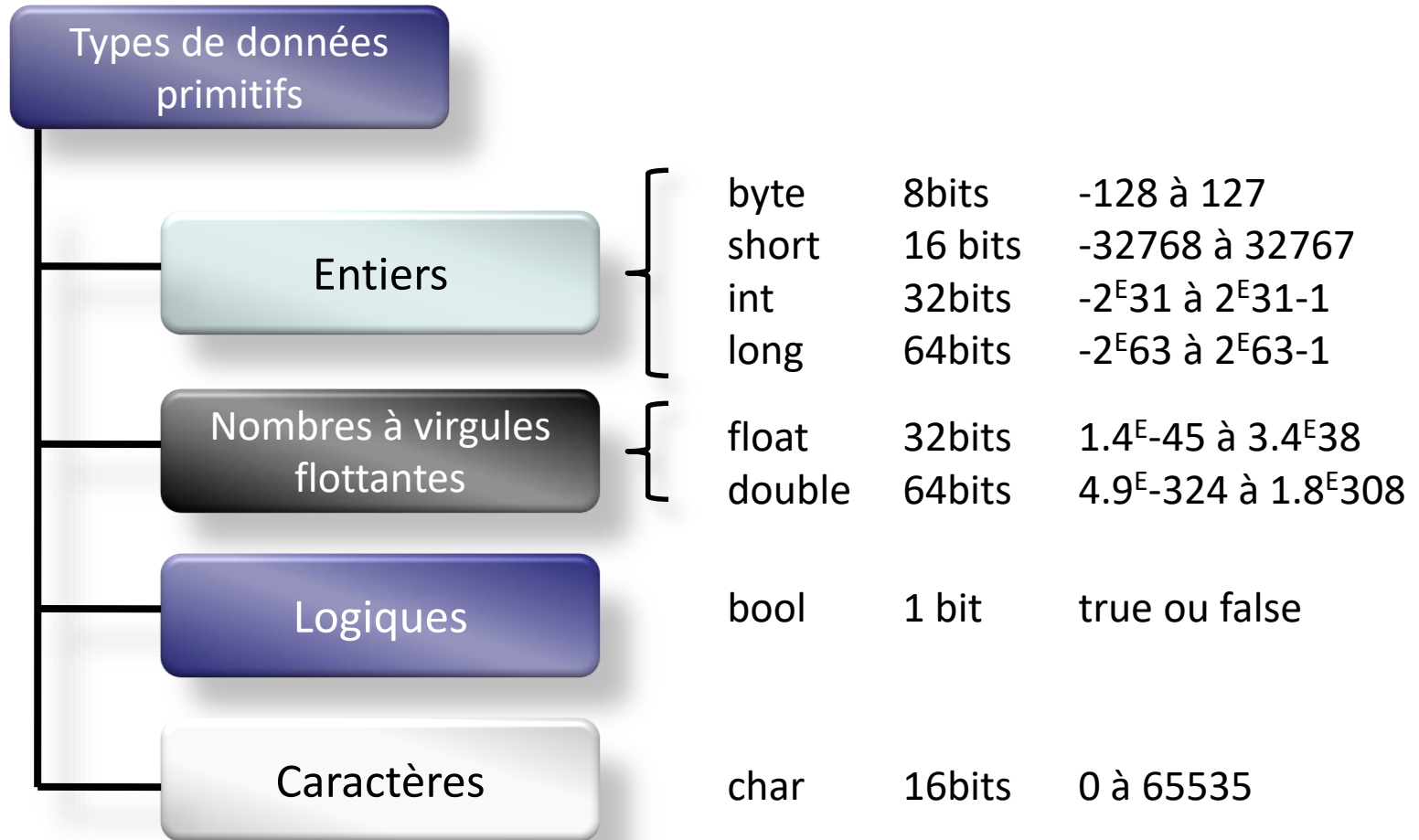
Le **type** d'une donnée en Java précise les **valeurs** qu'elle peut contenir et les **opérations** que l'on peut réaliser dessus.

Il existe deux type de données:

- Types **primitifs** : contient **physiquement** la valeur. La déclaration d'un type primitif alloue un emplacement pour stocker une valeur du type considéré.
- Types de **références** (ou types objets) : stocke en **mémoire** l'adresse où l'information se trouve. La déclaration d'un type objet alloue donc un emplacement pour stocker une **référence** vers un objet.

Pour **déclarer** une **variable**, on précise son **type** et son **identificateur**.

III . Types primitifs et types de références



Exemples de déclaration:

```
int a;
```

```
float b;
```

III . Types primitifs et types de références

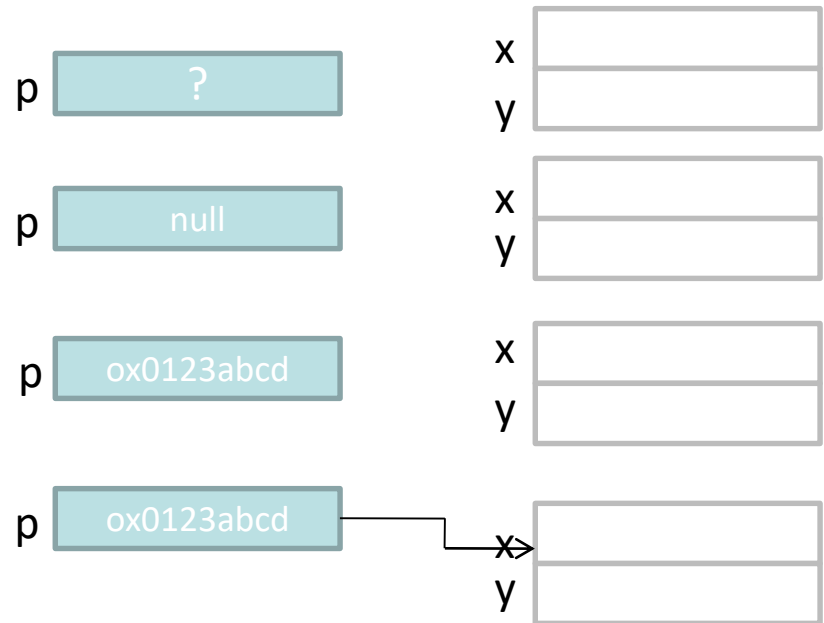
Types de références

Tous les types sauf les types primitifs. On les appelle aussi les **types objets**.

Exemple:

```
Point p;  
p = new Point(2,3);
```

- 1 – recherche une place en mémoire
- 2 – assignation d'une valeur par défaut
- 3 – appel au constructeur de la classe
- 4 – création de la référence/du pointeur



Aperçu du chapitre

- I. **Commenter son code source**
- II. **Les mots-clés (mots réservés) et les identificateurs en Java**
- III. **Types primitifs et types de références**
- IV. Arithmétique et opérateurs**
- V. **Expressions, instructions et blocs**
- VI. **Instruction de branchement et de contrôle**
- VII. **Les tableaux et la classe String**

IV . Arithmétique et opérateurs – *Opérateurs mathématiques et règle de précedence sur les opérateurs*

En Java , les **opérateurs mathématiques** existants sont repris dans le tableau suivant:

Niveau de priorité	Symbole	Signification
1	()	Parenthèse
2	* / %	Produit Division Modulo
3	+ -	Addition Soustraction

IV . Arithmétique et opérateurs – *Opérateurs d'affectation*

En Java , **l'opérateur d'affectation** est symbolisé par un simple = :
variable = expression;

Cette instruction signifie que la variable (destination) reçoit la valeur de l'expression (source).

La **valeur initiale** de la variable est **écrasée** par l'opérateur =.

Il est impossible de faire apparaître une opération comme premier opérande le l'opérateur :

b = a + 3;	➔ ok
a + 3 = b;	➔ impossible

Les affectations multiples sont autorisées.

i = j = k = 10;

IV . Arithmétique et opérateurs – *Opérateurs d'affectation*

Il existe également des opérateurs d'affectation réalisant en même temps une opération arithmétique ou logique.

Ceux-ci sont repris dans le tableau suivant:

Opérateur	Exemple	Equivalent à
+=	a += b expr1 += expr2	a = a + b expr1 = expr1 + expr2
-=	a -= b expr1 -= expr2	a = a - b expr1 = expr1 - expr2
*=	a *= b expr1 *= expr2	a = a * b expr1 = expr1 * expr2
/=	a /= b expr1 /= expr2	a = a / b expr1 = expr1 / expr2
%=	a %= b expr1 %= expr2	a = a % b expr1 = expr1 % expr2

IV . Arithmétique et opérateurs – *Le transtypage*

Les **opérateurs arithmétiques** ne sont définis que lorsque deux **opérandes** sont de **mêmes types**. Cependant, il est fréquent d'écrire des expressions mixtes... (additionner un int et un double, par exemple).

Le **transtypage** (ou **cast**) est la **conversion** d'une expression d'un certain type en une expression d'un autre type.

Le transtypage peut être :

- **Implicite** : on peut affecter à un champ ou une variable d'un type donné une expression de type moins élevé dans la hiérarchie des types. Cette transformation est accompagnée par un **gain de précision**.
- **Explicite** : on utilise la syntaxe suivante – *(nouveau_type)expression* – à chaque fois que l'on souhaite convertir une expression dans un type qui n'est pas plus haut dans la hiérarchie des types. Cette transformation est accompagnée par une **perte de précision**.

IV . Arithmétique et opérateurs – *Le transtypage*

La hiérarchie des types primitifs est la suivante :

byte < short/char < int < long < float < double

Exemple de transtypage implicite avec les types primitifs :

```
int n;  
float f;  
n = 3;  
f = n;
```

=> f vaudra 3.0

Exemple de transtypage explicite avec les types primitifs :

```
int n;  
float f;  
f = 3.8;  
n = (int) f;
```

=> n vaudra 3

IV . Arithmétique et opérateurs – *Opérateurs d'incrémentation et de décrémentation*

Très utilisé, les **opérateurs d'incrémentation et de décrémentation** sont les suivants :

Opérateur	Exemple	Equivalent à
++	i++	i = i + 1 i += 1
--	i--	i = i - 1 i -= 1

IV . Arithmétique et opérateurs – *Opérateurs de comparaison de valeurs*

En Java, les **opérateurs de comparaisons de valeurs** existants sont repris dans le tableau suivant:

Opérateur	Exemple	Renvoi <i>true</i> si
> >=	a > b a >=b	a plus grand que b a plus grand ou égal à b
< <=	a < b a <=b	a plus petit que b a plus petit ou égal à b
== !=	a == b a != b	a égal b a différent de b

IV . Arithmétique et opérateurs – *Opérateurs de logique*

En Java, les **opérateurs de logique** existants sont repris dans le tableau suivant:

Opérateur	Exemple	Renvoi <i>true</i> si
&&	expr1 && expr2	expr1 et expr2 sont vraies
	expr1 expr2	expr1 ou expr2 ou les deux sont vraies
!	!expr1	expr1 est fausse

IV . Arithmétique et opérateurs – *Opérateur conditionnel*

L'opérateur ternaire **?:** permet de **réaliser un test**.

Syntaxe :

(expression_booléenne) ? expression1 : expression2

Si l'expression booléenne est vraie, l'expression1 est évaluée, sinon c'est l'expression2 qui est évaluée.

Le **résultat** de l'opérateur est l'**expression évaluée**.

Expression1 et expression2 doivent **retourner le même type**.

Exemple :

```
int max = (a > b) ? a : b;
```

IV . Arithmétique et opérateurs – Exercices

Quels sont les valeurs des variables a, b, c et d après l'exécution de chacun des extraits de programmes suivants :

1)

```
float a = 3.5, b = 1.5, c;
```

```
c = a + b;
```

```
b = a + c;
```

```
a = b;
```

2)

```
int a = 2, b = 7;
```

```
a = a + 1;
```

```
a = a * 2;
```

```
a = a % 5;
```

```
b = a;
```

```
a = b;
```

IV . Arithmétique et opérateurs – Exercices

Soit 2 variables a et b. Quelle solution utiliser pour échanger leurs valeurs ?

a = 1; b = 5; ➔ a = 5; b = 1;

Donner les valeurs des expressions suivantes en sachant que *i* et *j* sont de type *int* et *x* et *y* sont de types *double* (*x* = 2.0, *y* = 3.0)

- a) $i = 100/6;$
- b) $j = 100 \% 6;$
- c) $i = 5 \% 8;$
- d) $(3 * i - 2 * j) / (2 * x - y);$
- e) $2 * ((i / 5) + (4 * (j - 3)) \% (i + j - 2));$
- f) $(i - 3 * j) / (x + 2 * y) / (i - j)$

IV . Arithmétique et opérateurs – Exercices

Donner le type et la valeur des expressions suivantes, sachant que n , p , r , s et t sont de type *int* ($n = 10$, $p = 7$, $r = 8$, $s = 7$, $t = 21$) et que x est de type *double* ($x = 2.0$)

- a) $x + n \% p$
- b) $x + n / p$
- c) $(x + n) / p$
- d) $5. * n$
- e) $(n + 1) / n$
- f) $(n + 1.0) / n$
- g) $r + s / t$
- h) $r + t / s$
- i) $(r + t) / s$
- j) $r + t \% s$
- k) $(r + t) \% s$
- l) $r + s / r + s$
- m) $(r + s) / (r + s)$
- n) $r + s \% t$

IV . Arithmétique et opérateurs – Exercices

Soit les déclarations suivantes :

int valeur = 7, chiffre = 2, i1, i2;

double d1, d2;

Quelles sont les valeurs attribuées après ces calculs :

- a) $i1 = \text{valeur} / \text{chiffre};$
- b) $i2 = \text{chiffre} / \text{valeur};$
- c) $d1 = (\text{double}) (\text{valeur} / \text{chiffre})$
- d) $d2 = (\text{double}) (\text{valeur} / \text{chiffre}) + 0.5;$
- e) $i1 = (\text{int}) d1;$
- f) $i2 = (\text{int}) d2;$
- g) $d1 = (\text{double}) \text{valeur} / (\text{double}) \text{chiffre};$
- h) $d2 = (\text{double}) \text{valeur} / (\text{double}) \text{chiffre} + 0.5;$
- i) $i1 = (\text{int}) d1;$
- j) $i2 = (\text{int}) d2;$

Aperçu du chapitre

- I. **Commenter son code source**
- II. **Les mots-clés (mots réservés) et les identificateurs en Java**
- III. **Types primitifs et types de références**
- IV. **Arithmétique et opérateurs**
- V. Expressions, instructions et blocs**
- VI. **Instruction de branchement et de contrôle**
- VII. **Les tableaux et la classe String**

V. Expressions, instructions et blocs

En Java, une **expression** permet de calculer et d'assigner des valeurs.

Une expression **réalise un calcul** sur des éléments et **retourne un résultat**.

Exemple :

$$a + 1$$
$$(b/4) + 6\%(c+1)$$

On peut construire des expressions complexes à partir d'expressions plus simples à conditions que les types imposé par les opérateurs soient respectés.

~~"Hello World!" + 6%44~~

V. Expressions, instructions et blocs

En Java, les **instructions** sont comparables aux phrases du langage courant.

Une instruction constitue l'unité d'exécution.

Exemples:

```
int a = 3;  
System.out.println(« Hello World! »);  
b = 6%3*(3+2);
```

En Java, les instructions sont **toujours** terminées par un ;

V. Expressions, instructions et blocs

En Java, les **blocs** sont une **suite d'instructions** regroupées entre deux accolades.

Si elles permettent de regrouper « logiquement » des instructions entres-elles, elles ne sont pas obligatoires.

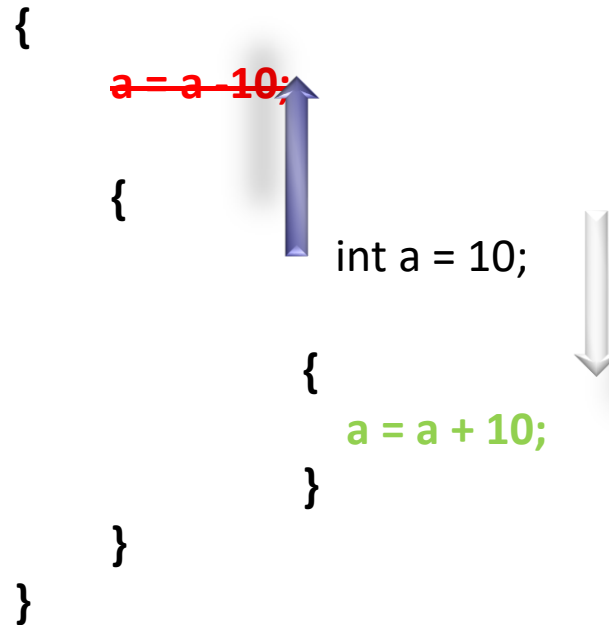
Elles le deviennent par contre lors de l'utilisation, par exemple, de structures de contrôle ou la déclaration de méthodes.

```
{  
    vitesse = vitesse + 10;  
    vitesse += 30;  
    System.out.println(vitesse);  
}
```

V. Expressions, instructions et blocs

Les blocs permettent d'introduire la notion de **portée d'une variable**:

Cette notion veut qu'une **variable** déclarée à l'intérieur d'un bloc ne soit **connue** que de ce bloc et des autres blocs qu'il pourrait contenir.



Aperçu du chapitre

- I. **Commenter son code source**
- II. **Les mots-clés (mots réservés) et les identificateurs en Java**
- III. **Types primitifs et types de références**
- IV. **Arithmétique et opérateurs**
- V. **Expressions, instructions et blocs**
- VI. Instruction de branchement et de contrôle**
- VII. **Les tableaux et la classe String**

VI. Instruction de branchement et de contrôle

En Java, les **instructions de branchement et de contrôle** permettent d'arrêter l'exécution linéaire (de haut en bas et de gauche à droite) des instructions d'un programme.

Elles permettent d'exécuter **conditionnellement** des instructions ou de réaliser des **boucles**.

Type d'instruction	Mots clés
Décision	if (...) else – switch() case
Boucle	for(;;) – while() – do while()
Traitement d'exceptions	try catch finally – throw
Branchement	label – break – continue - return

VI. Instruction de branchement et de contrôle – *if* – *if else* – *if else if else*

```
if (expression est vraie)
{
    instruction 1;
    instruction 2;
}
```

Si l'expression est vraie alors les instructions entre accolades sont exécutées.

```
if (expression est vraie)
{
    instruction 1;
    instruction 2;
}
else
{
    instruction 3;
    instruction 4;
}
```

Si l'expression est vraie alors les instructions entre accolades sont exécutées.

Autrement, ce sont les instructions du else qui sont exécutées.

VI. Instruction de branchement et de contrôle – *if* – *if else* – *if else if else*

```
if (expression est vrai)
```

```
{
```

```
    instruction 1;  
    instruction 2;
```

```
}
```

```
else if (expression est  
vrai)
```

```
{
```

```
    instruction 3;  
    instruction 4;
```

```
}
```

```
else
```

```
{
```

```
    instruction 5;  
    instruction 6;
```

```
}
```

Si l'expression est vraie alors les instructions entre accolades sont exécutées.

Autrement, ce sont les instructions du else if qui sont exécutées si et seulement si l'expression est vrai.


Si aucune des deux expressions n'est vraie, on tombe dans le else qui est alors une sorte de choix par défaut.

VI. Instruction de branchement et de contrôle – *switch case*

```
switch (expression)
{
case 1:  instruction 1;
        instruction 2;
        break;

case 2:  instruction 3;
        instruction 4;
        break;

default: instruction 5;
        instruction 6;
}
```



Structure "*atteindre ... cas x ... cas y ...*" :
embranchement vers un bloc d'instructions
énumérées.

« expression » peut être de type **byte**, **short**,
int, **char** ou **String**

VI. Instruction de branchement et de contrôle - Exercices

1. Ecrivez et exécutez un programme Java capable de générer un nombre aléatoire de type double (compris entre 0 et 1), de déterminer dans quel quintile d'intervalle il se trouve et de renvoyer cette information. Un quintile est l'une des cinq sections égales qui constituent le tout. Il s'agit des intervalles qui vont de 0 à 1/5, de 1/5 à 2/5, de 2/5 à 3/5, de 3/5 à 4/5 et de 4/5 à 1.
2. Ecrivez et exécutez un programme Java capable de générer un entier aléatoire et d'indiquer s'il est divisible par 2, 3 ou 5.
3. Ecrivez et exécutez un programme Java capable de générer un entier aléatoire compris dans l'intervalle 40 à 99, puis d'imprimer une lettre d'évaluation correspondant à l'appréciation de cette note dans le cadre d'un test (<60 : F, <70 : D, <80 : C, <90 : B, >=90 : A). Le signe « + » sera utilisé pour les notes se terminant par 8 ou 9 et le « - » pour celles qui se terminent par 0 ou 1. Par exemple, 78 correspond à un « C+ » et 90 à un « A- ».
4. Ecrivez et exécutez un programme Java capable de simuler une machine à calculer dont les opérations sont +, -, * et /.

VI. Instruction de branchement et de contrôle – *for*

```
for (expression d'initialisation ; expression de poursuite ; expression  
d'incrémentation)
```

```
{
```

```
    instruction 1;
```

```
    instruction 2;
```

```
    instruction 3;
```

```
}
```

Exemple:

```
for (int i = 1; i <= 10 ; i = i +1)
```

```
{
```

```
    System.out.println("la boucle tourne  pour la" + i + " fois");
```

```
}
```

VI. Instruction de branchement et de contrôle – *while*

```
while (expression est vrai)
{
    instruction 1;
    instruction 2;
    instruction 3;
}
```

Exemple:

```
boolean repeat = true;
while (repeat)
{
    ...
    if (...) {
        repeat = false;
    }
}
```

VI. Instruction de branchement et de contrôle – *do while*

```
do
{
    instruction 1;
    instruction 2;
}
while (expression est vrai)
```

Exemple:

```
boolean repeat = true;
do
{
    ...
    if (...) {
        repeat = false;
    }
}
while (repeat);
```


VI. Instruction de branchement et de contrôle – *break continue*

break : termine immédiatement la boucle même si celle-ci n'a pas fini de s'exécuter

continue : ignore la suite des instructions et reprend au début de la boucle

Exemple:

```
for (int i = 1; i <= 10 ; i = i +1)
{
    if (i == 5) continue;
    if (i == 7) break;
    System.out.println("la boucle tourne pour la" + i + " fois");
}
```



Instructions ...

VI. Instruction de branchement et de contrôle – *break [label]*


break [label]: termine immédiatement la boucle même si celle-ci n'a pas fini de s'exécuter

Et se rend à l'endroit du code indiqué par *[label]*

Exemple:

outer:

```
for (int i = 0; i <= 10 ; i = i +1)
{
    for (int j = 20; i > 4 ; j = j -1)
    {
        if (i == 5) continue;
        if (i == 7) break outer;
        System.out.println("la valeur de i et de j" + i + " - " + j);
    }
}
```



Instructions ...

VI. Instruction de branchement et de contrôle - Exercices

1. Ecrivez et exécutez un programme Java capable de générer un entier aléatoire compris entre 0 et 20, de calculer sa factorielle et d'en imprimer le résultat. Exemple de factorielle: $5! = 1 * 2 * 3 * 4 * 5 = 120$ ($0! = 1$).
2. Ecrivez et exécutez un programme Java capable d'imprimer la moyenne de 5 entiers aléatoires (entre 0 et 100). La sortie doit être similaire à la suivante : moyenne = 58.32154854
3. Ecrivez et exécutez un programme Java capable de faire deviner à l'utilisateur un nombre compris entre 1 et 100. Après chaque tentative qui échoue, l'ordinateur aide l'utilisateur en lui indiquant si le nombre proposé était trop petit ou trop grand (utiliser l'opérateur ternaire). Après chaque tentative, affichez également le nombre d'essais restants (nombre maximum d'essais : 5).
4. Ecrivez et exécutez un programme Java capable d'afficher l'ensemble des nombres premiers inférieurs ou égal à un nombre entier positif entré par l'utilisateur.

VI. Instruction de branchement et de contrôle - Exercices

5. Créer un ding-ding bottle.

Consignes:

Ecrire les nombres de 1 à 100 en :

- Remplaçant par « ding-ding », les nombres terminant par 5, les multiples de 5 ou les nombres dont la somme des chiffres qui le composent fait 5
- Remplaçant par « bottle », les nombres terminant par 7, les multiples de 7 ou les nombres dont la somme des chiffres qui le composent fait 7
- Remplaçant par « ding-ding bottle », les nombres remplissant les 2 critères ci-dessus

VI. Instruction de branchement et de contrôle - Exercices

6. Créer un programme qui génère un nombre aléatoire compris entre 2 et 20. En imprimer ensuite la séquence de Fibonacci.

Exemple pour 7 (le 0 n'est pas obligatoire) :

0, 1, 1, 2, 3, 5, 8, 13

7. Créer un programme capable de générer 8 triplets de Pythagore.

Exemple :

1) 20, 21, 29	$400 + 441 = 841$
2) 12, 35, 37	$144 + 1225 = 1369$
3) 48, 14, 50	$2304 + 196 = 2500$
4) 48, 20, 52	$2304 + 400 = 2704$
5) 12, 35, 37	$144 + 1225 = 1369$
6) 30, 72, 78	$900 + 5184 = 6084$
7) 4, 3, 5	$16 + 9 = 25$
8) 12, 16, 20	$144 + 256 = 400$

Nombre total d'essais : 85277

Aperçu du chapitre

- I. Commenter son code source**
- II. Les mots-clés (mots réservés) et les identificateurs en Java**
- III. Types primitifs et types de références**
- IV. Arithmétique et opérateurs**
- V. Expressions, instructions et blocs**
- VI. Instruction de branchement et de contrôle**
- VII. Les tableaux et la classe String**

VII. Les tableaux et la classe String – *Les tableaux à 1 et n dimensions*

Il existe deux grandes catégories de tableaux:

- Les tableaux à **une** dimension (ou **vecteurs**):

Ceux-ci peuvent-être imaginés comme les wagons d'un train.

5	6	7	85	0	6	3
---	---	---	----	---	---	---

- Les tableaux à **n** dimensions (ou **matrices**):

Ceux-ci peuvent-être imaginés comme une grille de combat naval (2 dimensions) ou encore un rubik's cube (3 dimensions).

A	R	5
K	5	6
6	3	P



VII. Les tableaux et la classe String – *Les tableaux à 1 dimension*

Un **tableau** est utilisé pour stocker un **ensemble**, une collection de **variables** de **même type**.

Un tableau est un **type objet**.

La **création** d'un tableau nécessite **deux informations** :

- le type de données qu'il va contenir
- sa taille

On peut créer des tableaux de types primitifs ou de types objets.

Une fois un tableau créé, on ne pourra **plus modifier sa taille**.

Le champ **length** permet de retourner la taille d'un tableau :
nomTableau.length

VII. Les tableaux et la classe String – *Les tableaux à 1 dimension*

Un tableau à **une** dimension en Java est obtenu de la manière suivante:

- **Déclaration** : `int [] tableauEntiers;`
- **Création**: `tableauEntiers = new int[10];`
ou
- **Déclaration et création** : `int[] tableauEntiers = new int[10];`

Si le tableau est connu à l'avance, on peut utiliser la syntaxe suivante :

```
int[] tableauEntiers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Pour **accéder** à un **élément** du tableau, on utilise son **indice** :

```
tableauEntiers[3] = 4;
```

```
int a = tableauEntiers[3] + 10;
```

Les **indices** d'un tableau sont des **nombres entiers positifs** et **commencent** obligatoirement par **0** et non par 1.

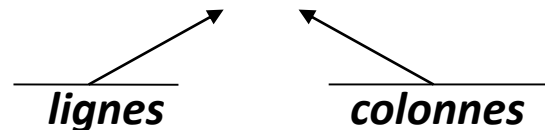
VII. Les tableaux et la classe String – *Les tableaux à n dimensions*

Dans la mesure où on peut déclarer des tableaux de n'importe quel type, on peut déclarer des tableaux de tableaux (ou des tableaux de tableaux de tableaux, etc.).

Un tableau à **n** dimensions en Java est obtenu de la manière suivante:

- **Déclaration** : `int [] [] matrice_entiers;`
- **Création**: `matrice_entiers = new int[10] [5];`

ou



- **Déclaration et création** : `int[] [] matrice_entiers = new int[10] [5];`

VII. Les tableaux et la classe String – *Les tableaux à n dimensions*

Dans un tableau à **n** dimensions, on n'est pas obligé de préciser la valeur de la dernière dimension:

```
int[] [] matrice_entiers = new int[10] [];  
matrice_entiers[0] = new int [3];  
matrice_entiers[1] = new int [7];  
...
```

VII. Les tableaux et la classe String – *La copie de tableau*

Prenons le code ci-dessous :

```
int[] origine = { 11, 22, 33 };  
int[] cible = { 1, 2, 3 };  
cible = origine;  
cible[0] = 5;
```

Après exécution, **cible[0]** et **origine[0]** auront pour valeur 5.

Pourquoi?

Parce que l'instruction **cible = origine;** a copié la référence de **origine** dans celle de **cible**.

Pour copier efficacement un tableau, il faut utiliser la méthode **Arrays.copyOf** :

```
int[] origine = { 11, 22, 33 };  
int[] cible = { 1, 2, 3 };  
cible = Arrays.copyOf(origine, origine.length);  
cible[0] = 5;
```


VII. Les tableaux et la classe String – *La classe String*

En Java, les **chaînes de caractères** ne sont pas des types primitifs, mais des types objets représentés par la classe **String**.

La classe String a la particularité d'avoir une notation pour les constantes littérales :

String chaine = « Ma chaine »;

Les chaînes de caractères sont entourées par des **doubles guillemets**:

" **Hello World!** "

Attention:

Si les chaînes de caractères sont entourées par des doubles guillemets, les caractères (le type char) sont eux entourés par de simples guillemets:

' **C** '

VII. Les tableaux et la classe String – Exercices

1. Ecrivez et testez un programme qui permet d'afficher tous les éléments d'un tableau (quelle que soit sa taille). Demandez à l'utilisateur d'encoder la taille du tableau et de remplir les éléments du tableau.
2. Ecrivez et testez un programme capable de fusionner deux tableaux d'entiers triés pour en créer un troisième qui sera trié lui aussi. Les entiers placés dans les tableaux doivent être choisis aléatoirement.
3. Ecrivez et testez un programme capable de dire si un tableau est trié ou non. S'il est trié, précisez l'ordre (croissant ou décroissant).
4. Ecrivez et testez un programme qui remplit un tableau $n * n$ avec les tables de multiplications jusqu'à n

Exemple : $3 * 3$

1	2	3
2	4	6
3	6	9

VII. Les tableaux et la classe String – Exercices

5. Ecrivez et testez un programme qui calcule le triangle de Pascal jusqu'à la 9^e ligne. Pour cela, vous devez remplir un tableau avec les valeurs adéquates puis afficher uniquement les valeurs utiles.

1									
1	1								
1	2	1							
1	3	3	1						
1	4	6	4	1					
1	5	10	10	5	1				
1	6	15	20	15	6	1			
1	7	21	35	35	21	7	1		
1	8	28	56	70	56	28	8	1	

Exercices BONUS

1. L'utilisateur encode la hauteur d'un triangle isocèle. Dessiner le triangle à l'aide du caractère « * ».

```
Encodez la hauteur du triangle : 5
  *
 ***
*****
*****
*****
```

2. L'utilisateur encode un mot et le programme lui dit s'il s'agit ou non d'un palindrome (ex: kayak).
3. L'utilisateur encode une phrase. Le programme affiche le nombre d'occurrences de chaque caractère présent dans la phrase

```
Encodez une phrase:
Check! Check! Check! This is a test!
Liste des caractères:
Le nombre de   est de 6
Le nombre de ! est de 4
Le nombre de C est de 3
Le nombre de T est de 1
Le nombre de a est de 1
Le nombre de c est de 3
Le nombre de e est de 4
Le nombre de h est de 4
Le nombre de i est de 2
Le nombre de k est de 3
Le nombre de s est de 3
Le nombre de t est de 2
```