



# MINDS' ACADEMY

Niveau	3A & 3B
Chapiter 1	Introduction au langage de programmation Java
Objectif	<ul style="list-style-type: none"><li>• familiariser l'étudiant avec la programmation élémentaire en Java: types, les énoncés de contrôle, les boucles, les tableaux...</li><li>• préparer l'étudiant à suivre un cours plus avancé</li></ul>

# Table des matières

## 1 Introuction au langage Java

1.1 Qu'est-ce que Java.....	3
1.2 Pourquoi apprendre le langage Java?.....	3
1.2.1 La simplicité.....	3
1.2.2 Une très grande communauté.....	3
1.2.3 Un langage typé.....	4
1.3 Environnement Java.....	4
1.3.1 Compilateur .....	5
1.3.2 Interprétation .....	5

## 2 Syntaxe de base

2.1 Règles générales d'écriture.....	6
2.1.2 Les identificateurs.....	6
2.1.3 Commentaire.....	7
2.2 Les types et les variables.....	7
2.2.3 Les types primitifs.....	7
2.2.4 Les types par référence.....	8

## 3 Éléments de programmation Java

3.1 Premier exemple de programme Java.....	9
3.2 Tableaux et matrices.....	10
3.3 Chaînes de caractères.....	11
3.4 Opérateurs.....	12
3.4.1 Opérateurs arithmétiques.....	12
3.4.2 Opérateurs entiers sur les bits.....	12
3.4.3 Opérateurs relationnels.....	12
3.4.4 Opérateurs booléens logiques.....	13
3.5 Les structures de contrôle.....	13
3.5.1 Structures conditionnelles .....	13
3.5.2 La structure choix multiples.....	14
3.5.3 Structures itératives.....	15
3.6 Instructions break et continue.....	16

# 1 Introduction au langage Java

## 1.1 Qu'est-ce que Java

---

Java est un langage de programmation moderne développé par Sun Microsystems en 1995 (racheté par Oracle en 2010). Il permet une programmation orientée-objet, modulaire et reprend une syntaxe très proche de celle du langage C. Il ne faut surtout pas le confondre avec JavaScript (langage de scripts utilisé principalement sur les sites web), car Java n'a rien à voir. Une de ses plus grandes forces est son excellente portabilité : une fois votre programme créé, il fonctionnera automatiquement sous Windows, Mac, Linux, etc. En contrepartie, les applications Java ont le défaut d'être plus lentes à l'exécution que des applications programmées en C par exemple

## 1.2 Pourquoi apprendre le langage Java?

---

Voici les quelques raisons pour lesquelles nous pensons que Java est le bon langage de programmation pour commencer à apprendre à coder :

### 1.2.1 La simplicité

Pour éviter de perdre votre motivation et votre amour pour la programmation, il est fortement recommandé de débiter avec un langage simple. C'est le cas de Java qui est à la fois facile à comprendre et en plus offre une très grande lisibilité. Il est idéal pour les novices étant donné qu'il leur permet d'avancer rapidement tout en assimilant aisément les différentes étapes du codage.

### 1.2.2 Une très grande communauté

Java est soutenu par une très grande communauté, et peu importe le type de questions, de doutes ou de problèmes que vous avez, Google peut vous aider à trouver des réponses. Et si ce n'est pas Google, alors Stack Overflow, les forums Java et beaucoup d'autres communautés sont là pour vous aider. C'est vraiment la raison principale pour laquelle nous suggérons aux débutants d'apprendre à coder en utilisant Java

### 1.2.3 Un langage typé

Java est un langage fortement typé qui décèle beaucoup d'erreurs de débutants. Ce langage convient aussi dans une moindre mesure pour la dactylographie statique. C'est une autre raison pour laquelle nous suggérons aux débutants d'apprendre d'abord Java, puis Python parce que Python est un langage à typage dynamique. Vous n'avez pas besoin de définir des types qui rendent l'apprentissage un peu plus difficile.

## 1.3 Environnement Java

Java est un langage interprété, ce qui signifie qu'un programme compilé n'est pas directement exécutable par le système d'exploitation mais il doit être interprété par un autre programme, qu'on appelle interpréteur. La figure 1.1 illustre ce fonctionnement.

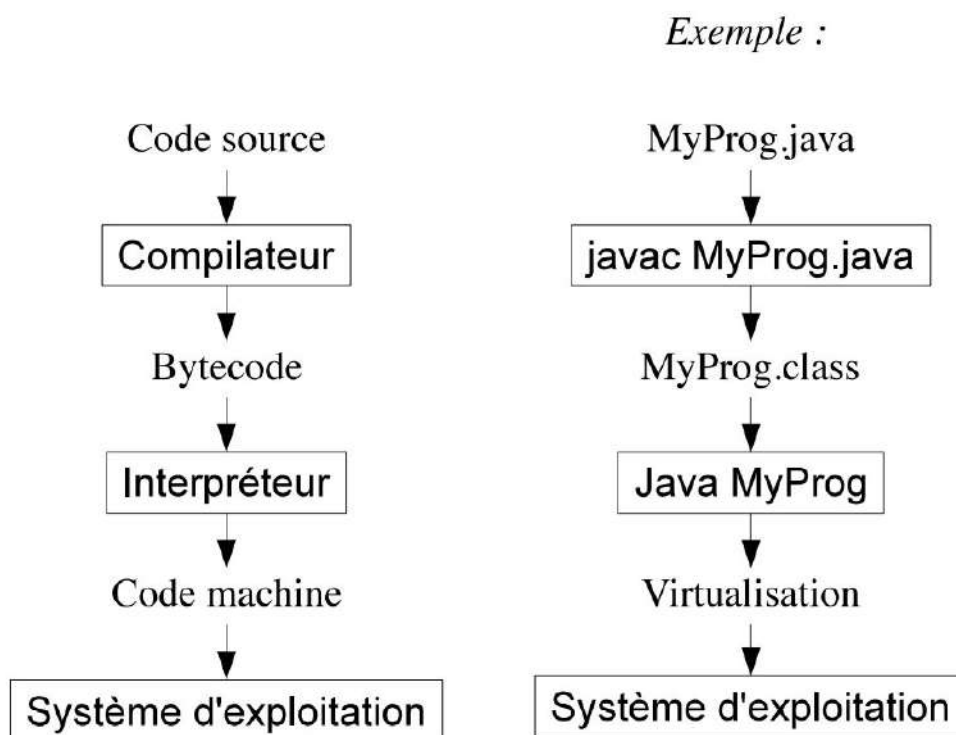


FIGURE 1.1 – Interprétation du langage

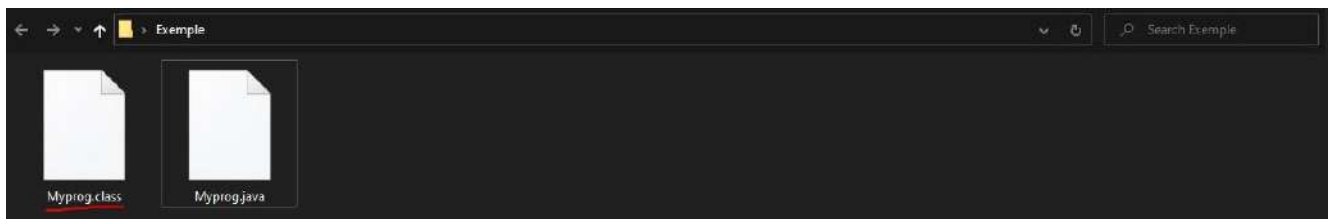
### 1.3.1 Compilateur

La compilation s'effectue par la commande `javac` suivie d'un ou plusieurs nom de fichiers contenant le code source de classes Java. Par exemple, `javac MyProg.java` compile la classe `MyProg` dont le code source est situé dans le fichier `MyProg.java`. Le résultat de cette compilation est un fichier nommé `MyProg.class` contenant le bytecode correspondant au source compilé. Ce fichier est créé par défaut dans le répertoire où la compilation s'est produite.

### 1.3.2 Interprétation

Le bytecode obtenu par compilation ne peut être exécuté qu'à l'aide de l'interpréteur. L'exécution s'effectue par la commande `java` suivie du nom de la classe à exécuter (sans l'extension `.class`)

```
C:\Users\Mortadha\Desktop\Exemple>javac Myprog.java
```



```
C:\Users\Mortadha\Desktop\Exemple>java Myprog
Hello

C:\Users\Mortadha\Desktop\Exemple>
```

# 2 Syntaxe de base

## 2.1 Règles générales d'écriture

Voici quelques connaissances de base:

- Les blocs de code sont encadrés par des accolades
- Chaque instruction se termine par un ";"
- Une instruction peut tenir sur plusieurs lignes.
- L'indentation (la tabulation) est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur.

### 2.1.2 Les identificateurs

Chaque objet, classe, programme ou variable est associé à un nom : l'identificateur qui peut se composer de tous les caractères alphanumériques et des caractères "\_" et "\$". Le premier caractère doit être une lettre, le caractère de soulignement ou le signe "\$".

**Remarque :** depuis Java 9, l'identifiant composé uniquement d'un caractère underscore n'est plus valide car \_ est devenu un mot clé du langage.

Un identifiant ne peut pas être: un mot clé du langage  
true ou false qui sont des booléens littéraux  
null qui est une valeur littérale

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

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0



**MINDS' ACADEMY**

### 2.1.3 Commentaire

Ils ne sont pas pris en compte par le compilateur donc ils ne sont pas inclus dans le pseudo code. Ils ne se terminent pas par un caractère ";". Il existe trois types de commentaire en Java :

- **commentaire abrégé:** // commentaire sur une seule ligne
- **commentaire multiligne:** /\* commentaires ligne 1  
commentaires ligne 2 \*/
- **commentaire de documentation automatique:** /\*\* documentation \*/

## 2.2 Les types et les variables

Java est un langage à typage fort. Lors de la déclaration d'une variable, il faut spécifier le type de cette variable. Toute variable de votre programme doit avoir un type et un nom exemple:

```
int n;
```

```
String chaine;
```

Il existe deux sortes de variables : Les types primitifs qui contiennent des valeurs élémentaires (i.e. de simples suites de bits dans la mémoire).

Exemple : int, float , boolean...

Les références aux objets qui contiennent des références aux objets. Exemple : String...

### 2.2.1 Les types primitifs

Il existe 8 types primitifs en Java de taille différente

Type primitif	Signification	Place occupée en mémoire
byte	Entier très court allant de -128 à +127	1 octet
short	Entier court allant de -32768 à +32767	2 octets
int	Entier allant de -2 147 483 648 à +2 147 483 647	4 octets
long	Entier long allant de $-2^{63}$ à $+2^{63} - 1$	8 octets
float	Nombre réel allant de $-1.4 * 10^{-45}$ à $+3.4 * 10^{38}$	4 octets
double	Nombre réel double précision allant de $4.9 * 10^{-324}$ à $+1.7 * 10^{308}$	8 octets
char	Caractère unicode (65536 caractères possibles)	2 octets
boolean	variable booléenne (valeurs : vrai ou faux)	1 octet



### 2.2.2 Les types par référence

En fait, ce sont de variables destinées à contenir des adresses d'objets ou de tableaux, ce que l'on exprime par : les données non primitives sont manipulées "par référence" alors que les données primitives sont manipulées par valeur.

Il est important de comprendre cette manipulation par référence des types non primitifs. Par exemple, si on appelle une méthode en lui envoyant en paramètre un tableau, on transmet en fait (comme en C) l'adresse du tableau ; si la méthode appelée modifie le contenu d'un tableau reçu en paramètre, elle modifie aussi le contenu du tableau dont on lui a transmis l'adresse, puisqu'il s'agit du même tableau ! De même pour les attributs des objets transmis en paramètres lors d'appels de méthodes. Signalons que la valeur par défaut de tout attribut d'un type "par référence" est la valeur null.





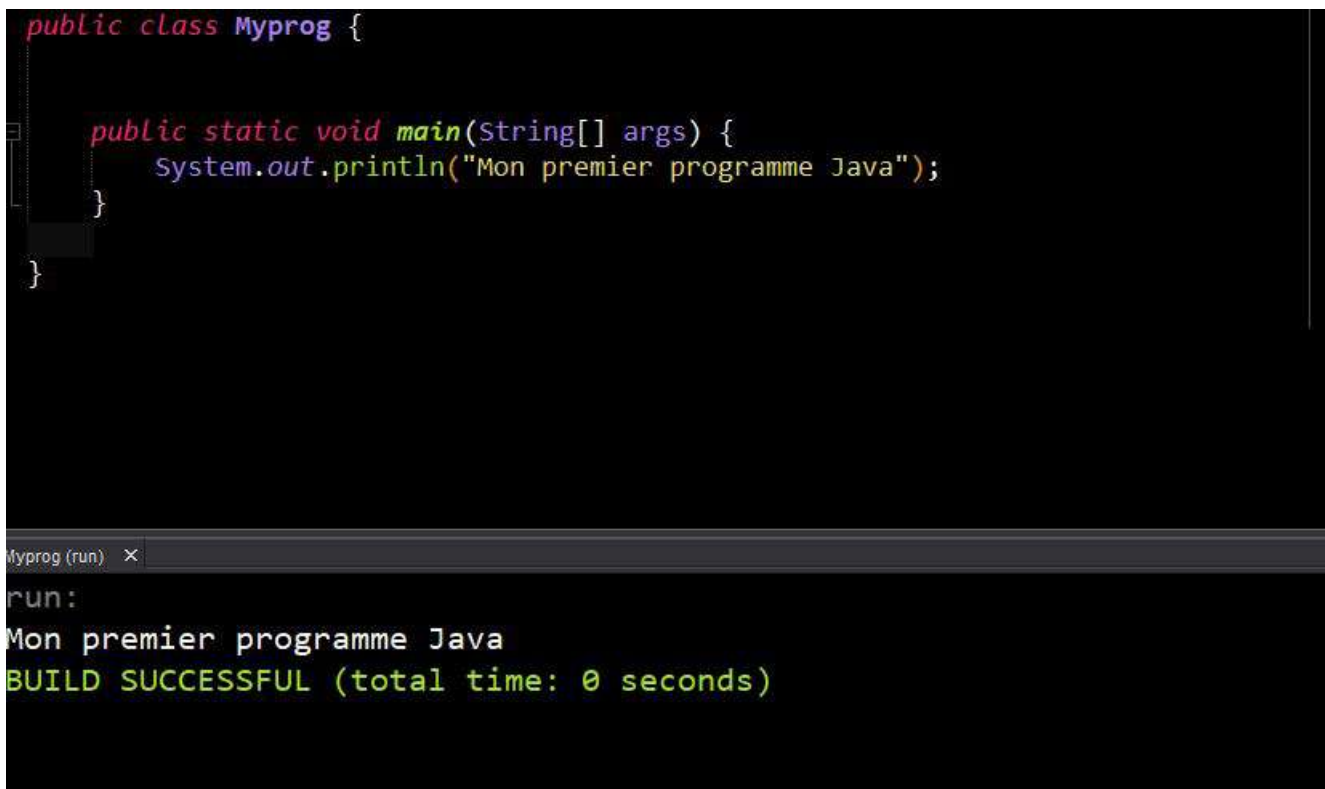
# 3 Éléments de programmation Java

Cette partie constitue une première approche d'un programme Java, fondée sur quelques exemples commentés. Vous y verrez, de manière informelle pour l'instant, comment s'expriment les instructions de base (déclaration, affectation, écriture...), ainsi que deux structures fondamentales (boucle avec compteur et choix).

## 3.1 Premier exemple de programme Java

---

Voici un exemple très simple de programme qui se contente d'afficher dans la fenêtre console le texte : "Mon premier programme Java"



```
public class Myprog {  
  
    public static void main(String[] args) {  
        System.out.println("Mon premier programme Java");  
    }  
}
```

Myprog (run) x

run:  
Mon premier programme Java  
BUILD SUCCESSFUL (total time: 0 seconds)

Vous constatez que, globalement, sa structure se présente ainsi: `public class Myprog { ..... }` Elle correspond théoriquement à la définition d'une classe nommée `Myprog`. La première ligne identifie cette classe ; elle est suivie d'un bloc, c'est-à-dire d'instructions délimitées par des accolades `{` et `}` qui définissent le contenu de cette classe. Ici, cette dernière est réduite à la seule définition d'une "méthode" particulière nommée `main` : `public static void main (String [] args) { System.out.println ("Mon premier programme Java") ; }` Là encore, une première ligne identifie la méthode ; elle est suivie d'un bloc (`{ ..... }`) qui en fournit les différentes instructions.

Pour l'instant, vous pouvez vous contenter d'utiliser un tel canevas, sans vraiment connaître les notions de classe et de méthode.

Il vous suffit simplement de placer dans le bloc le plus interne les instructions de votre choix, comme vous le feriez dans le programme principal (ou la fonction principale) d'un autre langage. Simplement, afin d'utiliser dès maintenant le vocabulaire approprié, nous parlerons de la méthode main de notre programme formé ici d'une seule classe nommée PremProg.

## 3.2 Tableaux et matrices

---

Une variable est déclarée comme un tableau dès lors que des crochets sont présents soit après son type, soit après son identificateur. Les deux syntaxes suivantes sont acceptées pour déclarer un tableau d'entiers

```
int[] tableau ;
int tableau2[];
```

Un tableau a toujours une taille fixe qui doit être précisée avant l'affectation de valeurs à ses indices, de la manière suivante :

```
int[] tableau=new int[20];
```

De plus, la taille de ce tableau est disponible dans une variable length appartenant au tableau et accessible par tableau.length. On peut également créer des matrices ou des tableaux à plusieurs dimensions en multipliant les crochets

(ex : int[ ][ ] matrice;). À l'instar du C, on accède aux éléments d'un tableau en précisant un indice entre crochets (tableau[3] est le quatrième entier du tableau) et un tableau de taille n stocke ses éléments à des indices allant de 0 à n-1.



### 3.3 Chaînes de caractères

Les chaînes de caractères ne sont pas considérées en Java comme un type primitif ou comme un tableau. On utilise une classe particulière, nommée `String`, fournie dans le package `java.lang`. Les variables de type `String` ont les caractéristiques suivantes :

- leur valeur ne peut pas être modifiée
- on peut utiliser l'opérateur `+` pour concaténer deux chaînes de caractères

Exemple:

```
String s1 = "hello" ;
String s2 = "world" ;
String s3 = s1 + " " + s2 ;
//Après ces instructions s3 vaut "hello world"
//L'initialisation d'une chaîne de caractères s'écrit:
String s4 = new String(); //pour une chaîne vide
String s5 = new String("hello world");
// pour une chaîne de valeur "hello world"
```

un ensemble de méthodes de la classe `java.lang.String` permettent d'effectuer des opérations ou des tests sur une chaîne de caractères par exemple:

```
15 public static void main(String[] args) {
16     String str="Minds Academy";
17     System.out.println("1)\n"+str.equals("Minds Academy"));
18     System.out.println("2)\n"+str.length());
19     System.out.println("3)\n"+str.charAt(0));
20     System.out.println("4)\n"+str.indexOf("d"));
21     System.out.println("5)\n"+str.toUpperCase());
22 }
```

```
Output - JavaApplication1 (run) X
run:
1)
true
2)
13
3)
M
4)
3
5)
MINDS ACADEMY
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 3.4 Opérateurs

### 3.4.1 Opérateurs arithmétiques

Op.	Résultat	Op.	Résultat
+	addition	+=	assignation additive
-	soustraction	-=	assignation soustractive
*	multiplication	*=	assignation multiplicative
/	division	/=	assignation divisionnelle
%	modulo	%=	assignation modulo
++	incrémentement	-	décrémentement

### 3.4.2 Opérateurs entiers sur les bits

Op.	Résultat	Op.	Résultat
-	NON unaire bit-à-bit		
&	ET bit-à-bit	&=	assignation avec ET bit-à-bit
	OU bit-à-bit	=	assignation avec OU bit-à-bit
^	OU exclusif bit-à-bit	^=	assignation avec OU exclusif bit-à-bit
>>	décalage à droite	>>=	assignation avec décalage à droite
>>>	décalage à droite <b>avec remplissage de zéros</b>	>>>=	assignation avec décalage à droite et remplissage de zéros
<<	décalage à gauche	<<=	assignation avec décalage à gauche

### 3.4.3 Opérateurs relationnels

Op.	Résultat
==	égal à
!=	différent de
>	strictement supérieur à
<	strictement inférieur à
>=	supérieur ou égal à
<=	inférieur ou égal à

### 3.4.4 Opérateurs booléens logiques

Op.	Résultat
&	ET logique
&=	assignation avec ET
	OU logique
=	assignation avec OU
^	OU exclusif logique
^=	assignation avec OU exclusif
	OU avec court circuit

## 3.5 Les structures de contrôle

Le principe d'un programme est de modifier le contenu des variables à l'aide des instructions élémentaires que nous venons de voir (affectation et opérateurs). Or, nous pouvons vouloir que ces instructions ne soient réalisées que dans certains cas, ou bien nous pouvons vouloir répéter l'exécution de ces instructions. Ce sont les structures de contrôle qui permettent de spécifier si l'exécution d'un traitement est conditionnée ou bien si elle se fait de manière répétée.

### 3.5.1 Structures conditionnelles

Les structures de contrôle conditionnelles permettent de spécifier à quelles conditions un bloc d'instructions va être exécuté. Cette condition est exprimée par une expression logique. LA STRUCTURE ALTERNATIVE Le premier type de conditionnelle s'écrit comme suit :

```

if (condition) { // équivalent à (condition == true)
    // bloc d'instructions exécutées si condition est vraie
} else {
    // bloc d'instructions exécutées si condition est fausse
}

```

**Exemple:**

```

int time = 20;
if (time < 18) {
    System.out.println("Good day.");
} else {
    System.out.println("Good evening.");
}

```

**3.5.2 La structure choix multiples**

Le second type de conditionnelle permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Sa syntaxe est la suivante :

```

switch (variable) {
    case valeur1 :
        Liste d'instructions // exécutées si (variable == valeur1) break;
    case valeur2 :
        Liste d'instructions // exécutées si (variable == valeur2) break;
    ...
    case valeurN :
        Liste d'instructions // exécutées si (variable == valeurN)
        break;
    default:
        Liste d'instructions // exécutées sinon
}

```

**Exemple:**

```

switch (pièce)
{
    case 5 :
        // Compter les pièces de 5 cents
        break;
    case 10 :
        // Compter les pièces de 10 cents
        break ;
    case 20 :
        // Compter les pièces de 20 cents
        break ;
    default :
        System.out.println ("Piece impossible");
}

```



### 3.5.3 Structures itératives

Il existe 3 formes de structure itérative, chacune a un cadre d'utilisation bien spécifique que nous allons voir.

#### 1. L'itération répétée n fois

La première forme itérative est la boucle for. Elle permet de répéter un bloc d'instructions un nombre de fois fixé. Dans sa syntaxe, il faut déclarer et initialiser la variable qui sert de compteur de tours de boucle, indiquer la condition sur le compteur pour laquelle la boucle s'arrête et enfin donner l'instruction qui incrémente ou décrémente le compteur :

```
for (int compteur = 0 ; compteur < n ; compteur = compteur + 1) {  
    // bloc instructions répétées n fois  
}  
ou  
for (int compteur = n ; compteur > 0 ; compteur = compteur - 1) {  
    // bloc instructions répétées n fois  
}
```

#### 2. L'itération répétée tant qu'une condition est vraie

La seconde forme d'itérative est la boucle while. Elle exécute le bloc d'instructions tant que la condition est vraie. Le bloc peut ne jamais être exécuté. La syntaxe est la suivante :

```
while (condition) { // équivalent à (condition == true)  
    // bloc d'instructions répétées tant que condition est vraie.  
    // condition doit être modifiée dans ce bloc  
}
```

Cette structure exécute le bloc d'instructions tant que (while en anglais) la condition est réalisée. Il est important de toujours s'assurer que la condition deviendra fausse lors d'une itération de la structure itérative. Dans le cas contraire, l'exécution du programme ne s'arrêtera jamais.

Exemple:

```
int i = 0;  
while (i < 5) {  
    System.out.println(i);  
    i++;  
}
```





### 3. L'itération exécutée au moins une fois

La troisième forme d'itérative est la boucle "do while". C'est une variante de la boucle while, où la condition d'arrêt est testée après que les instructions ont été exécutées :

```
do {
    // bloc d'instructions exécutées
    // condition doit être modifiée dans ce bloc
}while (condition); // si condition est vraie,
                    // le bloc est exécuté à nouveau
```

Ne pas oublier le ; après la condition d'arrêt. Le bloc d'instructions est exécuté au moins une fois.

Exemple:

```
int i = 0;
do {
    System.out.println(i);
    i++;
}
while (i < 5);
```

### 3.6 Instructions break et continue

---

L'instruction break est utilisée pour sortir immédiatement d'un bloc d'instructions (sans traiter les instructions restantes dans ce bloc). Dans le cas d'une boucle on peut également utiliser l'instruction continue avec la différence suivante : break : l'exécution se poursuit après la boucle (comme si la condition d'arrêt devenait vraie) ; continue : l'exécution du bloc est arrêtée mais pas celle de la boucle. Une nouvelle itération du bloc commence si la condition d'arrêt est toujours vraie.

```
for (int i = 0, j = 0 ; i < 100 ; i++) {
    if (i > tab.length) {
        break ;
    }
    if (tab[i] == null) {
        continue ;
    }
    tab2[j] = tab[i];
    j++;
}
```

Des Cours Particuliers Sont toujours  
Disponible en JAVA .  
Pour S'inscrire Contactez nous sur :  
[Ouday.ba@outlook.fr](mailto:Ouday.ba@outlook.fr)



**MINDS' ACADEMY**