

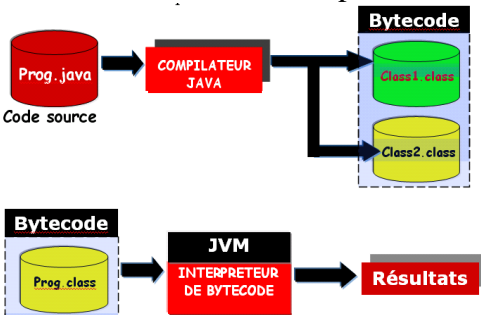
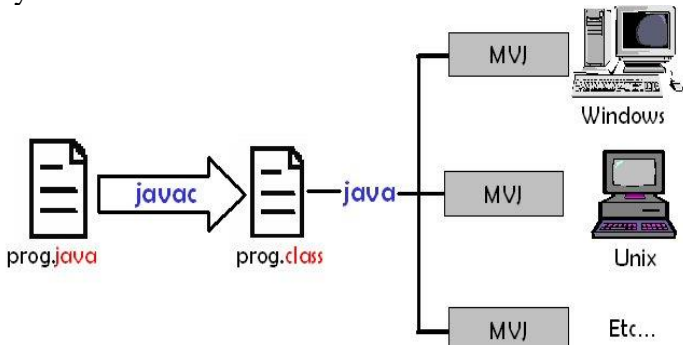
Les bases du langage java

1. Présentation

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C.

1.1. Les caractéristiques

Java possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès :

<p>Java est à la fois un langage compilé et interprété</p>	<p>Le programme source est compilé en pseudo code ou byte code puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout). En effet, le byte code, s'il ne contient pas de code spécifique à une plate-forme particulière peut être exécuté et obtenir quasiment les mêmes résultats sur toutes les machines disposant d'une JVM.</p>  <pre>graph LR A[Prog.java Code source] --> B[COMPILATEUR JAVA] B --> C[Bytecode Class1.class Class2.class] C --> D[JVM INTERPRETEUR DE BYTECODE] D --> E[Résultats]</pre>
<p>Java est portable : il est indépendant de toute plate-forme</p>	<p>Il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du byte code.</p>  <pre>graph LR A[prog.java] -- javac --> B[prog.class] B -- java --> C[MVJ Windows] B -- java --> D[MVJ Unix] B -- java --> E[MVJ Etc...]</pre>

	MVJ : Machine Virtuelle Java (=JVM aussi)
Java est orienté objet.	Comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).
Java est simple	<ul style="list-style-type: none"> • Il n'y a plus de pointeurs et des manipulations les concernant (pour éviter les incidents en manipulant directement la mémoire) ; • Java se charge (presque) de restituer au système les zones mémoire inaccessibles et ce sans l'intervention du programmeur.
Java est fortement typé	Toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.
Java assure la gestion de la mémoire	L'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.
Java est économe	Le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.
Java est multitâche	Il permet l'utilisation de threads qui sont des unités d'exécution isolées. La JVM, elle même, utilise plusieurs threads.

1.2. Les différentes éditions et versions de Java

Sun fournit gratuitement un ensemble d'outils et d'API pour permettre le développement de programmes avec Java. Ce kit, nommé JDK, est librement téléchargeable sur le site web de Sun <http://java.sun.com/> ou par FTP <ftp://java.sun.com/pub/>

Le JRE (Java Runtime Environment) contient uniquement l'environnement d'exécution de programmes Java. Le JDK contient lui même le JRE. Le JRE seul doit être installé sur les machines où des applications Java doivent être exécutées.

Depuis sa version 1.2, Java a été renommé Java 2. Les numéros de version 1.2 et 2 désignent donc la même version. Le JDK a été renommé J2SDK (Java 2 Software Development Kit) mais la dénomination JDK reste encore largement utilisée, à tel point que la dénomination JDK est reprise dans la version 5.0. Le JRE a été renommé J2RE (Java 2 Runtime Environment).

Trois éditions de Java existent :

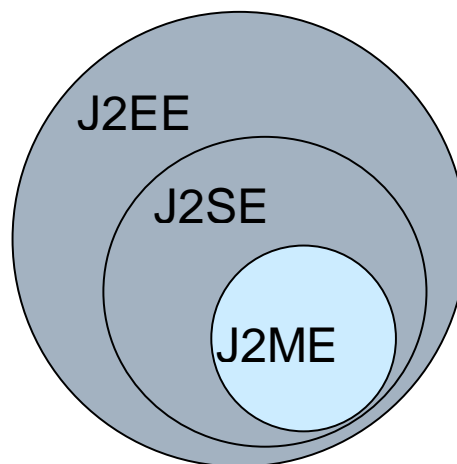
1. Java 2 Platform, Standard Edition (J2SE)
 - Applications "stand-alone"
2. Java 2 Platform, Enterprise Edition (J2EE)
 - Applications à large échelle, applications distribuées, applications Web, ...
 - Infrastructure « serveur » pour le support d'applications d'entreprise
 - Support des architectures multi-tiers
 - Architecture client léger (architecture Web basée « browser »)
 - Architecture client lourd
 - Architecture orientée service (application répartie sans présentation)

- La Java EE définit une architecture standard incluant
 - Un modèle de programmation (application multi-tiers, client légers)
 - Une plate-forme (ensemble de spécifications et de politiques requises)
 - Un ensemble de test de compatibilité
 - Une implantation de référence
 - Les standards de la Java EE sont gérés par la communauté **JCP** (Java Community Process www.jcp.org) à travers des JSR(Java Specification Request)
3. Java 2 Platform, Micro Edition (J2ME)
- Application pour terminaux mobiles (PDA, téléphones mobiles, ...)
 - J2ME™ est une spécification qui vise à définir une infrastructure logicielle pouvant s'adapter au développement/déploiement logiciel sur tous type de matériel mobile.
 - La J2ME peut être considérée comme une collection de classes et d'interfaces Java dédiés aux équipements mobiles (small and mobile devices).

Chaque édition définit un ensemble de bibliothèques de classes différent

Les équipements mobiles/embarqués qui sont basés sur utilisent peu de classes. Alors que les classes du runtime J2SE se comptent en milliers et occupent un espace allant vers 10-20 Mo.

Les trois éditions de Java exploitent les mêmes facilités syntaxiques du langage Java.



1.3. Les concepts de base

La plate-forme Java utilise quelques notions de base dans sa mise en oeuvre notamment :

- La compilation du code source dans un langage indépendant de la plate-forme d'exécution : le byte code
- l'exécution du byte code par une machine virtuelle nommée JVM (Java Virtual Machine)
- la notion de package qui permet d'organiser les classes
- le classpath qui permet de préciser au compilateur et à la JVM où elle peut trouver les classes requises par l'application
- le packaging de classes compilées dans une archive de déploiement nommé jar (Java ARchive)

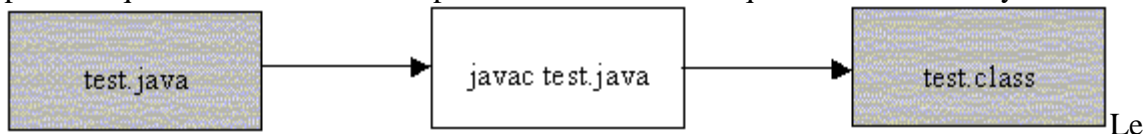
1.4. La compilation et l'exécution

Un programme Java est composé d'un ou plus généralement plusieurs fichiers source. N'importe quel éditeur de texte peut être utilisé pour éditer un fichier source Java.

Ces fichiers source possèdent l'extension .java. Ils peuvent contenir une ou plusieurs classes ou interfaces mais il ne peut y avoir qu'une seule classe ou interface déclarée publique par fichier. Le nom de ce fichier source doit obligatoirement correspondre à la casse prêt au nom de cette entité publique suivi de l'extension .java

Il est nécessaire de compiler le source pour le transformer en J-code ou byte-code Java qui sera lui exécuté par la machine virtuelle. Pour être compilé, le programme doit être enregistré au format de caractères Unicode : une conversion automatique est faite par le JDK si nécessaire.

Un compilateur Java, par exemple l'outil javac fourni avec le JDK est utilisé pour compiler chaque fichier source en fichier de classe possédant l'extension .class. Cette compilation gère pour chaque fichier source un ou plusieurs fichiers .class qui contiennent du byte code.



Le compilateur génère autant de fichier .class que de classes et interfaces définies dans chaque fichier source.

Pour exécuter une application, la classe servant de point d'entrée doit obligatoirement contenir une méthode ayant la signature **public static void main(String[] args)**. Il est alors possible de fournir cette classe à la JVM qui va charger le ou les fichiers .class utiles à l'application et exécuter le code.

Exemple :

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

Résultat :

```
C:\TEMP>javac MaClasse.java
```

```
C:\TEMP>dir MaClas*
```

```
MaClasse.class
```

```
MaClasse.java
```

Le compilateur génère autant de fichier .class que de classes et interfaces définies dans chaque fichier source.

Pour exécuter une application, la classe servant de point d'entrée doit obligatoirement contenir une méthode ayant la signature **public static void main(String[] args)**. Il est alors possible de fournir cette classe à la JVM qui va charger le ou les fichiers .class utiles à l'application et exécuter le code.

Exemple :

```
C:\TEMP>java MaClasse
```

```
Bonjour
```

3. La syntaxe et les éléments de bases de Java

Dans cette partie, nous allons traiter des aspects élémentaires du langage Java : les variables, les instructions de condition, etc. Nous n'aborderons pas encore les notions « objets ». Pour cette raison, tous les exemples seront écrits dans un « programme Java » basique.

3.1. Les règles de base

- Java est sensible à la casse (fait la différence entre la minuscule et la majuscule).
- Chaque instruction se termine par le caractère ';' (point virgule).
- Les blocs de code sont encadrés par des accolades.

3.2. 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 dollar.

Rappel : Java est sensible à la casse.

Il est important de donner des noms significatifs. La convention associée au langage Java est de coller les différents mots qui composent votre identificateur et d'indiquer chaque mot par une majuscule.

Par exemple si vous avez une variable qui vous sert pour la moyenne d'une classe, cela donnera *moyenneDeLaClasse*. La variable commence toujours par une minuscule.

Il est vraiment très important de donner des noms de variable qui ont un sens. Lft, ghy, b, z ne veulent rien dire et contribue à rendre votre application inutilisable par vous dans 6 mois ou par un partenaire de votre groupe.

Un identificateur ne peut pas appartenir à la liste des mots réservés du langage Java :

abstract	const	final	int	public	throw
assert (Java 1.4)	continue	finally	interface	return	throws
boolean	default	float	long	short	transient
break	do	for	native	static	true
byte	double	goto	new	strictfp	try
case	else	if	null	super	void
catch	enum (Java 5)	implements	package	switch	volatile
char	extends	import	private	synchronized	while
class	false	instanceof	protected	this	

3.3. Les commentaires

- Tout programme (grand ou petit, simple ou complexe) contient (ou devrait contenir) des commentaires.
- Ils ont pour but d'expliquer :
 - Ce qu'est censé faire le programme,
 - Les conventions adoptées,
 - Tout autre information rendant le programme lisible à soi même et surtout à autrui.
- 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 ";".
- Java dispose de trois types de commentaires :

- Les commentaires multilignes : commencent par les caractères `/*` et se terminent par `*/`.
- Les commentaires lignes : débutent avec les symboles `//` et se terminent à la fin de la ligne.
- Les commentaires de type documentation : Ces commentaires, appelés aussi commentaires javadoc, servent à documenter les classes que l'on définit. Ils sont encadrés entre `/**` et `*/`. Java exige que ces commentaires figurent avant la définition de la classe, d'un membre de la classe ou d'un constructeur. Ces commentaires serviront à produire automatiquement (avec l'outil javadoc) la documentation sous forme HTML à l'image de la documentation officielle de SUN.

Type de commentaires	Exemple
commentaire ligne	<code>// Ce programme imprime la chaîne // de caractères " bonjour " à l'écran</code>
commentaire multiligne	<code>/* Ce programme imprime la chaîne de caractères "bonjour" à l'écran */</code>
commentaire de documentation automatique	<code>/** * commentaire de la methode * @param val la valeur a traiter * @since 1.0 * @return Rien * @deprecated Utiliser la nouvelle methode XXX */</code>

3.4. La déclaration et l'utilisation de variables

3.4.1. La déclaration de variables

Une variable possède un nom, un type et une valeur. La déclaration d'une variable doit donc contenir deux choses : un nom et le type de données qu'elle peut contenir.

Rappel : les noms de variables suivent les règles de définition des identificateurs (voir partie 3.2)

En java, les variables peuvent être définies/déclarées à n'importe quel endroit. Elles n'ont pas à être définies au début (d'une méthode, d'une classe) comme en C. Une variable est utilisable dans le bloc où elle est définie.

La déclaration d'une variable permet de réserver la mémoire pour en stocker la valeur.

Le type d'une variable peut être :

- soit un type élémentaire dit aussi type primitif déclaré sous la forme : **type_élémentaire variable**; Exemple : `long nombre`;
- soit une classe déclarée sous la forme : **classe variable** ; Exemple : `String chaine`;

Il est possible de définir plusieurs variables de même type en séparant chacune d'elles par une virgule.

Exemple :

```
int jour, mois, annee ;
```

Java est un langage à typage rigoureux qui ne possède pas de transtypage automatique lorsque ce transtypage risque de conduire à une perte d'information.

3.4.2. Les types élémentaires

Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution : c'est un des éléments qui permet à Java d'être indépendant de la plate-forme sur lequel le code s'exécute.

Les entiers

Type	Taille	Intervalle
byte	8 bits	-128 à 127
short	16 bits	-32768 à 32767
int	32 bits	-2147483648 à 2147483647
long	64 bits	-9223372036854775808 à 9223372036854775807

Les réels

Type	Taille	Intervalle
float	32 bits	3.40282347E+28 à 1.40239846E-45
double	64 bits	1.79769313486231570E+308 à 4.9406545841246544E-324

Les caractères

Type	Taille	Intervalle
char	16 bits	0 à 65535

Un caractère est codé sur 16 bits car il est conforme à la norme Unicode. Il doit être entouré par des apostrophes. Exemple : char code = 'D';

Les booléens

Type	Taille	Intervalle
boolean	1 bit	true ou false

Les types élémentaires commencent tous par une minuscule.

Remarque : Le type **chaîne de caractères** n'est pas réellement un type élémentaire car il s'agit d'une classe Java. Mais il est considéré comme tel car il est codé en dur dans la machine virtuelle. Le type à indiquer est ***String***.

3.4.3. L'initialisation des variables

```
int nombre; // déclaration
```

```
nombre = 100; //initialisation
```

```
int nombre = 100; //déclaration et initialisation
```

3.4.4. L'affectation

le signe = est l'opérateur d'affectation et s'utilise avec une expression de la forme : **variable = expression**.

Il existe des opérateurs qui permettent de simplifier l'écriture d'une opération d'affectation associée à un opérateur mathématique :

Opérateur	Exemple	Signification
=	a=10	équivalent à : a = 10

+=	a+=10	équivalent à : a = a + 10
-=	a-=	équivalent à : a = a - 10
=	a=	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	a%=10	reste de la division

Attention : Lors d'une opération sur des opérandes de types différents, le compilateur détermine le type du résultat en prenant le type le plus précis des opérandes. Par exemple, une multiplication d'une variable de type float avec une variable de type double donne un résultat de type double. Lors d'une opération entre un opérande entier et un flottant, le résultat est du type de l'opérande flottant.

3.4.5. Les comparaisons

Java propose des opérateurs pour toutes les comparaisons :

Opérateur	Exemple	Signification
>	a > 10	strictement supérieur
<	a < 10	strictement inférieur
>=	a >= 10	supérieur ou égal
<=	a <= 10	inférieur ou égal
==	a == 10	Egalité
!=	a != 10	différent de
&	a & b	ET binaire
^	a ^ b	OU exclusif binaire
	a b	OU binaire
&&	a && b	ET logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient fausse
	a b	OU logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient vraie
?:	a ? b : c	opérateur conditionnel : renvoie la valeur b ou c selon l'évaluation de l'expression a (si a alors b sinon c) : b et c doivent retourner le même type

3.5. Les opérations arithmétiques

Les opérateurs arithmétiques se notent : + (addition), - (soustraction), * (multiplication), / (division) et % (reste de la division). Ils peuvent se combiner à l'opérateur d'affectation

Exemple :

nombre += 10;

3.5.1. L'arithmétique entière

Pour les types numériques entiers, Java met en oeuvre une sorte de mécanisme de conversion implicite vers le type int appelée promotion entière. Ce mécanisme fait partie des règles mise en place pour renforcer la sécurité du code.

Exemple :

short x= 5 , y = 15;


```
x = x + y ; //erreur à la compilation
```

Incompatible type for =. Explicit cast needed to convert int to short.

```
x = x + y ; //erreur à la compilation
```

^

1 error

Les opérandes et le résultat de l'opération sont convertis en type int. Le résultat est affecté dans un type short : il y a donc risque de perte d'informations et donc erreur à la compilation est émise. Cette promotion évite un débordement de capacité sans que le programmeur soit pleinement conscient du risque : il est nécessaire, pour régler le problème, d'utiliser une conversion explicite ou cast

Exemple :

```
x = (short) ( x + y );
```

Il est nécessaire de mettre l'opération entre parenthèse pour que ce soit son résultat qui soit converti car le cast a une priorité plus forte que les opérateurs arithmétiques.

3.5.2. L'incrémentement et la décrémentation

Les opérateurs d'incrémentement et de décrémentation sont : $n++$ $++n$ $n--$ $--n$

Si l'opérateur est placé avant la variable (préfixé), la modification de la valeur est immédiate sinon la modification n'a lieu qu'à l'issue de l'exécution de la ligne d'instruction (postfixé)

L'opérateur ++ renvoie la valeur avant incrémentement s'il est postfixé, après incrémentement s'il est préfixé.

Exemple :

```
System.out.println(x++); // est équivalent à
```

```
System.out.println(x); x = x + 1;
```

```
System.out.println(++x); // est équivalent à
```

```
x = x + 1; System.out.println(x);
```

Exemple :

```
/* test sur les incrementations prefixees et postfixees */
```

```
class test4 {  
    public static void main (String args[]) {  
        int n1=0;  
        int n2=0;  
        System.out.println("n1 = " + n1 + " n2 = " + n2);  
        n1=n2++;  
        System.out.println("n1 = " + n1 + " n2 = " + n2);  
        n1=++n2;  
        System.out.println("n1 = " + n1 + " n2 = " + n2);  
        n1=n1++;    //attention  
  
        System.out.println("n1 = " + n1 + " n2 = " + n2);  
    }  
}
```

Résultat :

```
int n1=0;
int n2=0; // n1=0 n2=0
n1=n2++; // n1=0 n2=1
n1=++n2; // n1=2 n2=2
n1=n1++; // attention : n1 ne change pas de valeur
```

3.6. La priorité des opérateurs

Java définit les priorités dans les opérateurs comme suit (du plus prioritaire au moins prioritaire)

les parenthèses	()
les opérateurs d'incrémentation	++ --
les opérateurs de multiplication, division, et modulo	* / %
les opérateurs d'addition et soustraction	+ -
les opérateurs de décalage	<< >>
les opérateurs de comparaison	< > <= >=
les opérateurs d'égalité	== !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	
l'opérateur ET logique	&&
l'opérateur OU logique	
les opérateurs d'assignement	= += -=

Les parenthèses ayant une forte priorité, l'ordre d'interprétation des opérateurs peut être modifié par des parenthèses.

3.7. Les structures de contrôles

Java propose un ensemble d'instructions qui permettent d'organiser et de structurer les traitements. L'usage de ces instructions est similaire à celui rencontré dans leur équivalent dans d'autres langages.

3.7.1. Les branchements conditionnels

If then else

```
if (condition)
    { instruction(s) }
else if (condition)
    { instruction(s) }

    else {instruction(s)}
```

Exemple:

```
public class ifApplication1 {

public static void main(String[ ] args)
{
int i = 5 ;
if (i % 2 == 0 )
{
    System.out.println (i+ " est pair " );
}
else
{
    System.out.println (i +"est impair " );
}
}
}
```

Résultat d'affichage: 5 est impair

Attention : Le résultat d'évaluation de la condition est obligatoirement de type boolean.
Si l'on écrit if (i = 1), le compilateur détectera une erreur, car le type de la condition est int.

Switch case

```
switch (expression)
{
    case valeur1 :
        instr11;
        instr12;
        break;

    case ...
        break;

    default :...
```

On ne peut utiliser switch qu'avec des types primitifs d'une taille maximum de 32 bits (byte, short, int, char).

Si une instruction case ne contient pas de break alors les traitements associés à l'instruction case suivante sont exécutés.

Exemple :

```
public class switchApplication {

    public static void main(String[ ] args)
    {
        int i = 1,
switch (i)
```

```

{
    case 0 :
        System.out.println (" 0 " ); break;
    case 1 :
        System.out.println (" 1 " ); break;
    case 2 :
        System.out.println (" 2 " ); break;
    default :
        System.out.println (i); break;
}
}
}

```

3.7.2. Les boucles

While

```

while ( condition )
{
    ... // code a exécuter dans la boucle
}

```

Le code est exécuté tant que la condition est vraie. Si avant l'instruction while, la condition est fausse, alors le code de la boucle ne sera jamais exécuté

Ne pas mettre de ; après la condition sinon le corps de la boucle ne sera jamais exécuté

Exemple:

```

public class whileApplication {
    public static void main(String[ ] args)
    {
        int compteur =1; int max = 4 ; int somme = 0;
        while (compteur < max )
        {
            somme = somme + compteur ;
            compteur++;
        }
        System.out.println (somme );
    }
}

```

Do While

```

do {
    ...
}
while ( condition )

```

Cette boucle est au moins exécutée une fois quelque soit la valeur du booléen;

Exemple :

```

public class do_WhileApplication {

    public static void main(String[ ] args)
    {
        int compteur =1; int max = 4 ; int somme = 0;
        do

```

```

{
    somme = somme + compteur ;
    compteur++;
}
while (compteur < max ) ;
System.out.println (somme );
}
}

```

For

Le *for* est généralement réservé pour un nombre donné d'itérations (pour le parcours d'un tableau par exemple). Sa syntaxe en Java est la suivante :

```

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

```

Exemple :

```

for (i = 0 ; i < 10; i++ ) { ....}
for (int i = 0 ; i < 10; i++ ) { ....}
for ( ; ; ) { ... } // boucle infinie

```

L'*initialisation* peut contenir la définition/déclaration d'une variable ou simplement (si celle-ci a déjà été déclarée) l'affectation de la valeur de départ.

La *condition* indique l'arrêt de l'itération : si celle-ci est fausse la « boucle » s'arrête. La boucle continue tant que cette condition est vraie.

L'*itération* est l'instruction à effectuer à la fin de chaque itération. Cela correspond généralement à l'incrément de l'indice d'itération.

Il est possible d'inclure plusieurs traitements dans l'initialisation et la modification de la boucle : chacun des traitements doit être séparé par une virgule.

Exemple :

```

public class forApplication {

    public static void main(String[ ] args)
    {
        int somme = 0;
        for ( int compteur = 1, max = 4 ; compteur < max ; compteur++)
        {
            somme = somme + compteur;
        }
        System.out.println (somme );
    }
}

```

○ Fonctionnement :

- initialisation du compteur,
- comparaison avec max,
- réalisation des instructions,
- Incrément de compteur et on recommence.

3.8. Les tableaux

Ils sont dérivés de la classe Object : il faut utiliser des méthodes pour y accéder dont font partie des messages de la classe Object tel que equals() ou getClass().

Le premier élément d'un tableau possède l'indice 0.

3.8.1. La déclaration des tableaux

Java permet de placer les crochets après ou avant le nom du tableau dans la déclaration.

Exemple :

```
int tableau[] = new int[50]; // déclaration et allocation
```

```
int[] tableau = new int[50];
```

```
int tab[]; // déclaration
```

```
tab = new int[50]; //allocation
```

Java ne supporte pas directement les tableaux à plusieurs dimensions : il faut déclarer un tableau de tableau.

Exemple :

```
float tableau[][] = new float[10][10];
```

La taille des tableaux de la seconde dimension peut ne pas être identique pour chaque occurrence.

Exemple :

```
int dim1[][] = new int[3][];
```

```
dim1[0] = new int[4];
```

```
dim1[1] = new int[9];
```

```
dim1[2] = new int[2];
```

3.8.2. L'initialisation explicite d'un tableau

Exemple :

```
int tableau[5] = {10,20,30,40,50};
```

```
int tableau[3][2] = {{5,1},{6,2},{7,3}};
```

La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création.

Exemple :

```
int tableau[] = {10,20,30,40,50};
```

Le nombre d'élément de chaque ligne peut ne pas être identique :

Exemple :

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

3.8.3. Le parcours d'un tableau

Exemple :

```
for (int i = 0; i < tableau.length ; i++) { ... }
```

La variable length retourne le nombre d'éléments du tableau.

Pour passer un tableau à une méthode, il suffit de déclarer les paramètres dans l'en tête de la méthode

Exemple :

```
public void printArray(String texte[]){ ...
```

}

Les tableaux sont toujours transmis par référence puisque ce sont des objets.

Un accès à un élément d'un tableau qui dépasse sa capacité, lève une exception du type `java.lang.arrayIndexOutOfBoundsException`.

3.9. Les conversions de types

Il existe deux catégories de conversions possibles :

- **Conversions explicites** : celles faites sur une demande explicite par un programmeur. Elles visent à changer le type d'une donnée si besoin. Pour faire une conversion explicite, il suffit de préfixer l'opérande à convertir par le type choisi et d'encadrer le type choisi par des parenthèses. **Exemple : `double d = 2.5 ; long l = (long) d ;`**
- **Conversions implicites** : celles faites automatiquement par un compilateur :

- lors d'une affectation :

Exemple1 :

```
public class ProgrammeAffectation {
    public static void main(String[ ] args)
    {
        int i;
        short j = 2 ;
        i = j ; //conversion implicite
```

```
float k= 1.2 f ;
```

```
i = k; //erreur de compilation, la solution pour corriger : i = (int) k;
}
}
```

Exemple2 :

```
public class ProgrammeSoustraction {
    public static void main(String[ ] args)
    {
        int i;
        i = 'A'; //conversion implicite
        System.out.print( i ) ;
    }
}
```

Résultat d'affichage : 65

- lors d'une promotion arithmétique : Si un opérateur s'applique sur deux arguments de type différent un des deux arguments sera converti dans le type de l'autre.

Exemple :

```
public class ProgrammeSoustraction {

    public static void main(String[ ] args)
    {
        int i;
        i = 'A'-1; //conversion implicite
        System.out.print( i ) ;
```

```

    }
    }
    ○ lors d'un passage de paramètres (lors de l'invocation d'une méthode),
    Exemple:
    public class ProgrammeSoustraction {
    static short entierShort;
    static void Afficher( int entier )
    {
        System.out.print( entier ) ;
    }
    public static void main(String[ ] args)
    {
    entierShort = 2 ;
    Afficher(entierShort ) ; //invocation de la méthode afficher
    }
    }

```

Remarque : Il n'y a pas de conversion possible (implicite ou explicite) entre un type entier et le type boolean :

Les conversions de type se font par des méthodes. La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion de types élémentaires.

Classe	Role
String	pour les chaines de caractères Unicode
Integer	pour les valeurs entières (integer)
Long	pour les entiers long signés (long)
Float	pour les nombres à virgules flottante (float)
Double	pour les nombres à virgule flottante en double précision (double)

Les classes portent le même nom que le type élémentaire sur lequel elles reposent avec la première lettre en majuscule.

Ces classes contiennent généralement plusieurs constructeurs. Pour y accéder, il faut les instancier puisque ce sont des objets.

Exemple :

```
String montexte;
```

```
montexte = new String("test");
```

L'objet montexte permet d'accéder aux méthodes de la classe java.lang.String

3.9.1. La conversion d'un entier int en chaîne de caractère String

Exemple :

```
int i = 10;
```

```
String montexte = new String();
```

```
montexte =montexte.valueOf(i);
```

valueOf est également définie pour des arguments de type boolean, long, float, double et char

3.9.2. La conversion d'une chaîne de caractères String en entier int

Exemple :

```
String montexte = new String(" 10 ");
Integer monnombre=new Integer(montexte);
int i = monnombre.intValue(); //conversion d'Integer en int
```

3.9.3. La conversion d'un entier int en entier long

Exemple :

```
int i=10;
Integer monnombre=new Integer(i);
long j=monnombre.longValue();
```

3.10. La manipulation des chaînes de caractères

La définition d'un caractère se fait grâce au type char :

Exemple :

```
char touche = '%';
```

La définition d'une chaîne se fait grâce à l'objet String :

Exemple :

```
String texte = " bonjour ";
```

Les variables de type String sont des objets. Partout où des constantes chaînes de caractères figurent entre guillemets, le compilateur Java génère un objet de type String avec le contenu spécifié. Il est donc possible d'écrire :

```
String texte = " Java Java Java ".replace('a','o');
```

Les chaînes de caractères ne sont pas des tableaux : il faut utiliser les méthodes de la classe String d'un objet instancié pour effectuer des manipulations.

Il est impossible de modifier le contenu d'un objet String construit à partir d'une constante. Cependant, il est possible d'utiliser les méthodes qui renvoient une chaîne pour modifier le contenu de la chaîne.

Exemple :

```
String texte = " Java Java Java ";
texte = texte.replace('a','o');
```

Java ne fonctionne pas avec le jeu de caractères ASCII ou ANSI, mais avec Unicode (Universal Code). Ceci concerne les types char et les chaînes de caractères. Le jeu de caractères Unicode code un caractère sur plusieurs octets. Les caractères 0 à 255 correspondent exactement au jeu de caractères ASCII étendu.

3.10.1. Les caractères spéciaux dans les chaînes

Dans une chaîne de caractères, plusieurs caractères particuliers doivent être utilisés avec le caractère d'échappement \. Le tableau ci dessous recense les principaux caractères.

Caractères spéciaux	Affichage
\'	Apostrophe
\"	Guillemet
\\	anti slash
\t	Tabulation
\b	retour arrière (backspace)
\r	retour chariot

<code>\f</code>	saut de page (form feed)
<code>\n</code>	saut de ligne (newline)
<code>\Oddd</code>	caractère ASCII ddd (octal)
<code>\xdd</code>	caractère ASCII dd (hexadécimal)
<code>\udddd</code>	caractère Unicode dddd (hexadécimal)

3.10.2. L'addition de chaînes

L'opérateur `+` permet de concaténer plusieurs chaînes. Il est possible d'utiliser l'opérateur `+=`

Exemple :

```
String texte = " ";
texte += " Hello ";
texte += " World3 ";
```

Cet opérateur sert aussi à concaténer des chaînes avec tous les types de bases. La variable ou constante est alors convertie en chaîne et ajoutée à la précédente. La condition préalable est d'avoir au moins une chaîne dans l'expression sinon le `+` est évalué comme opérateur mathématique.

Exemple :

```
System.out.println(" La valeur de Pi est : "+Math.PI);
int duree = 121;
System.out.println(" durée = " +duree);
```

3.10.3. La comparaison de deux chaînes

Il faut utiliser la méthode `equals()`

Exemple :

```
String texte1 = " texte 1 ";
String texte2 = " texte 2 ";
if ( texte1.equals(texte2) )...
```

3.10.4. La détermination de la longueur d'une chaîne

La méthode `length()` permet de déterminer la longueur d'une chaîne.

Exemple :

```
String texte = " texte ";
int longueur = texte.length();
```

3.10.5. La modification de la casse d'une chaîne

Les méthodes Java `toUpperCase()` et `toLowerCase()` permettent respectivement d'obtenir une chaîne tout en majuscule ou tout en minuscule.

Exemple :

```
String texte = " texte ";
String textemaj = texte.toUpperCase();
```