Langage Java

Le langage Java est un langage orienté objetcréeen1995 par *Sun Microsystems.* C'est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C. Aujourd'hui, le langage Java est très largement utilisé à la fois sur le Web, les stations de travail, les téléphones et autres tablettes. Il appartient à Oracle.

1-Java est un langage semi compilé :

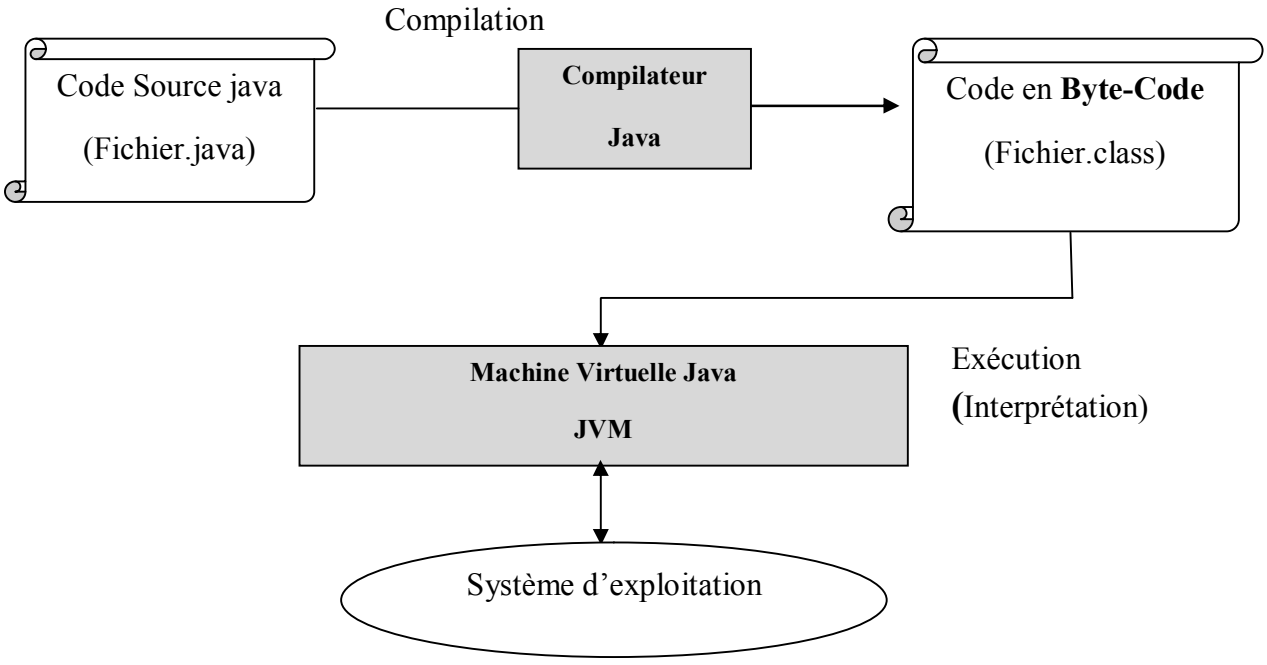
Java est un langage hybride, à la fois compilé et interprété. On dit qu'il est semi-compilé.Pour simplifier, disons qu'un programme Java est compilé dans un langage qui devra ensuiteêtre interprété. Le résultat de la compilation n'est pas du langage machine directementexécutable (propre au processeur), mais un code intermédiaire appelé **byte-code.**

Parmi les compilateurs java on site :

- JDK (Java Devlopper Kit) :
- Eclipse
- Netbeans
- JBulder
- Java Creator …

Le byte-codeest intermédiaire entre le code source et le langage machine.Pour exécuter le programme, le byte-code est interprété par un interpréteur appelémachine virtuelle Java (JVM).

Toutes les machines actuelles possèdent une JVM. Ainsi, lebyte-code d'un programme peut être exécuté sur n'importe quel ordinateur (possédant une JVM). C'est pour cela que le langage Java est un langage portable.



Les versions de Java pour les systèmes Windows, Solaris et Linux sont disponibles sur Internet. Chaque version de Java est fourniesous deux formes :

• L'une pour les développeurs : le **JDK** (Java Development Kit) ou SDK(Software Development Kit) comprenant la machine virtuelle Java pourun système d'exploitation, la bibliothèque des classes Java et les commandespour développer en Java(commande de compilation …).

• L'autre pour les utilisateurs : le**JRE** (Java Runtime Environment) comprenantla machine virtuelle Java et la bibliothèque des classes.

Exemple :

Si on utilisant le JDK avec la ligne de commande:

- Un programme écrit en java : MyProg.java doit être compilé par la commande **javac**fourni par le JDK comme suit :
  Javac MyProg.java

Cette dernière nous fournit un fichier bytecode c'est le fichier MyProg.class

- Ce fichier bytecode doit être exécuté (c'est le rôle JVM) en utilisant la commande **java**fourni par le JDK comme suit :
  Java MyProg.class

2- Autre Caractéristiques :

Simple

- Le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs (pour éviter les incidents en manipulant directement la mémoire), l'héritage multiple, ...
- Syntaxe proche du C et C++ : en reprenant une grande partie de la syntaxe de ces deux langages, Java facilite la formation initiale des programmeurs qui les connaissentdéjà.

Orienté objet

La principale caractéristique du langage Java est qu'il a été conçu directement comme un langage de POO contrairement au C++ où l'aspect objet n'est qu'une extension du langage C.

JAVA assure la gestion de mémoire

L'allocation de la mémoire pour un objet est automatique à sa création et Javarécupère automatiquement la mémoire inutilisée grâce au ***garbage collector*** quirestitue les zones de mémoire laissées libres suite à la destruction des objets.

Portable

Lebyte-code d'un programme peut être exécuté sur n'importe quel ordinateur possédant une JVM alors qu'un programme compilé en langage machine n'est exécutable que sur un seul type deprocesseur ou d'un système d'exploitation. C'est pour cela que le langage Java est un langage portable.

Java est distribué

- API réseau fournie en standard.
- Permettant de développer des Applets et des Servlets :
  - les applets : petit programme incorporé aux pages web et exécuté localement par lenavigateur du client.
  - les servlets : qui sont des programmes Java s'exécutant côté serveur et qui permettentde répondre à des requêtes http envoyées par un navigateur client.

- RMI (Remote Method Invocation).
- CORBA

Extensible avec Bibliothèque très riche : la bibliothèque fournie enstandard avec Javacouvre de nombreux domaines (gestionde collections, accès aux bases de données, interfaceutilisateur graphique, accès aux fichiers et auréseau, utilisation d'objets distribués, XML…) sanscompter toutes les extensions qui s'intègrent sans difficultéà Java.

Multitâche : grâce aux threads, Java permet de programmerl'exécution simultanée de plusieurs traitementset la synchronisation des traitements qui partagentdes informations.

3- Syntaxe java

3.1. Les règles de base

Java est sensible à la casse.

Les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ';' (point virgule).

Une instruction peut tenir sur plusieurs lignes :

L'indentation est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur.

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. Java est sensible à la casse.

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

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 :

Type de commentaires	Exemple
commentaire abrégé	`// commentaire sur une seule ligne` `int N=1; // déclaration du compteur`
commentaire multiligne	`/* commentaires ligne 1` `commentaires ligne 2 */`

commentaire de documentation automatique	`/**`
	`* commentaire de la methode`
	`* @param val la valeur à traiter`
	`* @since 1.0`
	`* @return la valeur de retour`
	`* @deprecated Utiliser la nouvelle methode XXX */`

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. 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;
- soit une classe déclarée sous la forme classe variable ;

Exemple :

```
1.longnombre;
2.intcompteur;
3.String chaine;
```

Les noms de variables en Java peuvent commencer par une lettre, par le caractère de soulignement ou par le signe dollar. Le reste du nom peut comporter des lettres ou des nombres mais jamais d'espaces.

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

Exemple :

```
intjour, mois, annee ;
```

Il est possible en une seule instruction de faire la déclaration et l'affectation d'une valeur à une variable ou plusieurs variables.

Exemple :

```
inti=3, j=4;
```

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 laquelle le code s'exécute.

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique : true ou false	1 bit	true ou false	pas de conversion possible vers un autre type
byte	octet signé	8 bits	-128 à 127	
short	entier court signé	16 bits	-32768 à 32767	
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans du code Java

int	entier signé	32 bits	-2147483648 à 2147483647	
long	entier long	64 bits	-9223372036854775808 à 9223372036854775807	
float	virgule flottante simple précision (IEEE754)	32 bits	1.401e-045 à 3.40282e+038	
double	virgule flottante double précision (IEEE754)	64 bits	2.22507e-308 à 1.79769e+308	

Les types élémentaires commencent tous par une minuscule.

3.4.3. Le format des types élémentaires

Le format des nombres entiers :

Il existe plusieurs formats pour les nombres entiers : les types byte, short, int et long peuvent être codés en décimal, hexadécimal ou octal. Pour un nombre hexadécimal, il suffit de préfixer sa valeur par 0x. Pour un nombre octal, le nombre doit commencer par un zéro. Le suffixe l ou L permet de spécifier que c'est un entier long.

Le format des nombres décimaux :

Il existe plusieurs formats pour les nombres décimaux. Les types float et double stockent des nombres flottants : pour être reconnus comme tels ils doivent posséder soit un point, un exposant ou l'un des suffixes f, F, d, D. Il est possible de préciser des nombres qui n'ont pas le partie entière ou pas de partie décimale.

Exemple :

```
1.floatpi = 3.141f;
2.doublevaleur = 3d;
3.floatflottant1 = +.1f , flottant2 = 1e10f;
```

Par défaut un littéral représentant une valeur décimale est de type double : pour définir un littéral représentant une valeur décimale de type float il faut le suffixer par la lettre f ou F.

Attention :

```
float pi = 3.141; // erreur à la compilation
float pi = 3.141f; // compilation sans erreur
```

Exemple :

```
1.doublevaleur = 1.1;
```

Le format des caractères :

Un caractère est codé sur 16 bis car il est conforme à la norme Unicode. Il doit être entouré par des apostrophes. Une valeur de type char peut être considérée comme un entier non négatif de 0 à 65535. Cependant la conversion implicite par affectation n'est pas possible.

Exemple :

```
1./* test sur les caractères */
2.classtest1 {
3.publicstaticvoidmain (String args[]) {
4.charcode = 'D';
5.intindex = code - 'A';
6.System.out.println("index = "+ index);
7.}
8.}
```

3.4.4. L'initialisation des variables

Exemple :

```
1.intnombre; // déclaration
2.nombre = 100; //initialisation
3.OU intnombre = 100; //déclaration et initialisation
```

En Java, toute variable appartenant à un objet (définie comme étant un attribut de l'objet) est initialisée avec une valeur par défaut en accord avec son type au moment de la création. Cette initialisation ne s'applique pas aux variables locales des méthodes de la classe.

Les valeurs par défaut lors de l'initialisation automatique des variables d'instances sont :

Type	Valeur par défaut
boolean	false
byte, short, int, long	0
float, double	0.0
char	\u000
classe	null

Remarque : Dans une applet, il est préférable de faire les déclarations et initialisations dans la méthode init().

3.4.5. L'affectation

Le signe = est l'opérateur d'affectation et s'utilise avec une expression de la forme variable = expression. L'opération d'affectation est associative de droite à gauche : il renvoie la valeur affectée ce qui permet d'écrire :

```
x = y = z = 0;
```

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-=10	équivalent à : a = a - 10
*=	a*=10	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	a%=10	reste de la division
^=	a^=10	équivalent à : a = a ^ 10
<<=	a<<=10	équivalent à : a = a << 10 a est complété par des zéros à droite
>>=	a>>=10	équivalent à : a = a >> 10 a est complété par des zéros à gauche
>>>=	a>>>=10	équivalent à : a = a >>> 10 décalage à gauche non signé

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.

3.4.6. 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	diffèrent 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

Les opérateurs sont exécutés dans l'ordre suivant à l'intérieur d'une expression qui est analysée de gauche à droite:

- incréments et décréments
- multiplication, division et reste de division (modulo)
- addition et soustraction
- comparaison
- le signe = d'affectation d'une valeur à une variable

L'usage des parenthèses permet de modifier cet ordre de priorité.

3.5L'incrémentation et la décrémentation

Les opérateurs d'incrémentation 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 incrémentation s'il est postfixé, après incrémentation s'il est préfixé.

Exemple :

```
1.System.out.println(x++); // est équivalent à
2.System.out.println(x); x = x + 1;
3.
4.System.out.println(++x); // est équivalent à
5.x = x + 1; System.out.println(x);
```

Exemple :

```
01./* test sur les incrementations prefixées et postfixées */
02.
03.classtest4 {
04.publicstaticvoidmain (String args[]) {
05.intn1=0;
06.intn2=0;
07.System.out.println("n1 = "+ n1 + " n2 = "+ n2);
08.n1=n2++;
09.System.out.println("n1 = "+ n1 + " n2 = "+ n2);
10.n1=++n2;
11.System.out.println("n1 = "+ n1 + " n2 = "+ n2);
12.n1=n1++;        //attention
13.System.out.println("n1 = "+ n1 + " n2 = "+ n2);
14.}
15.}
```

Résultat :

```
1.int n1=0;
2.int n2=0; // n1=0 n2=0
3.n1=n2++; // n1=0 n2=1
4.n1=++n2; // n1=2 n2=2
5.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

Comme la quasi-totalité des langages de développement orientés objets, 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 avec leur équivalent dans d'autres langages.

3.7.1. Les boucles

- Structure **tant que** (adaptée pour effectuer des opérations tant qu'une condition est remplie) :

```
while ( boolean )
{
    ... // code à exécuter dans la boucle
}
```

Le code est exécuté tant que le booléen est vrai. Si avant l'instruction while, le booléen est faux, 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

- Structure **faire … tant que** (comme la structure **tant que** mais la première itération est exécutée quelle que soit la condition, pour les autres itérations la condition doit être remplie) :

```
do {
   ...
} while ( boolean );
```

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

- Structure **pour**

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

Exemple :

```
1.for(i = 0; i < 10; i++ ) { ....}
2.for(inti = 0; i < 10; i++ ) { ....}
3.for( ; ; ) { ... } // boucle infinie
```

L'initialisation, la condition et la modification de l'index sont optionnelles.

Dans l'initialisation, on peut déclarer une variable qui servira d'index et qui sera dans ce cas locale à la boucle.

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 :

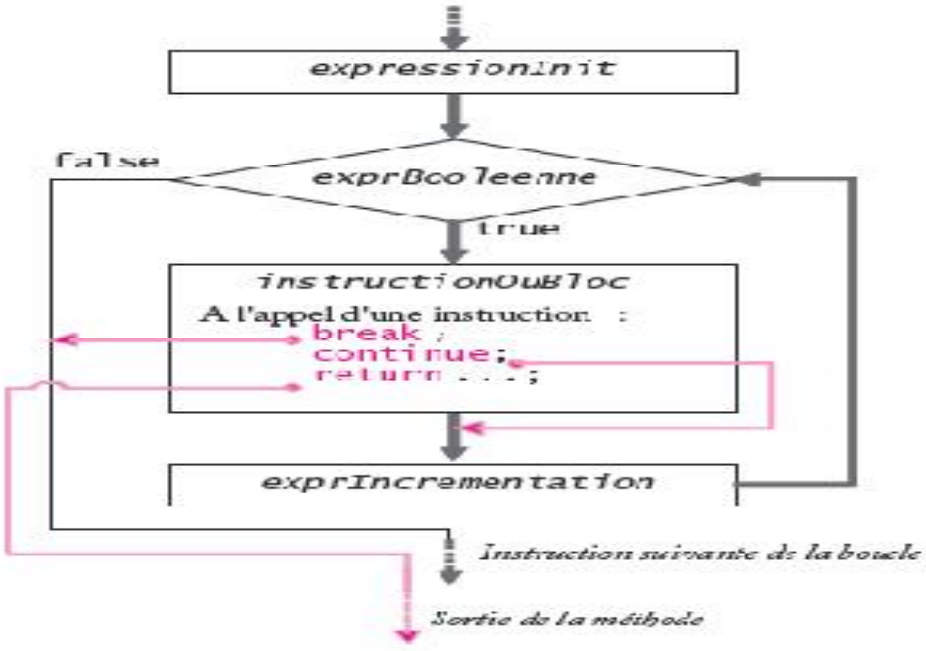
```
1.for(i = 0, j = 0; i * j < 1000;i++ , j+= 2) { ....}
```

La condition peut ne pas porter sur l'index de la boucle :

Exemple :

```
1.booleantrouve = false;
2.for(inti = 0; !trouve ; i++ ) {
3.if( tableau[i] == 1) {
4.trouve = true;
5.... //gestion de la fin du parcours du tableau
6.}
7.}
```

Voici la séquence d'exécution de la boucle for :



3.7.2. Les branchements conditionnels

Les instructions de branchement if et switch déterminent le traitement à effectuer quand une condition est vérifiée ou non.

- Structure **si** : condition simple

```
if (<expression booléenne>) {
   instruction(s)
}
```

- Structure **si … sinon** : condition avec alternative unique

```
if (<expression booléenne>) {
     instruction(s)
}
 else {
     instruction(s)
 }
```

- Structure **si … ou si … ou si …** : condition avec alternatives multiples

```
if (<expression booléenne>) {
     instruction(s)
 }
 else if (<expression booléenne>) {
     instruction(s)
 }
```

```
 else if (<expression booléenne>) {
     instruction(s)
}
 else {
     instruction(s)
 }
if (boolean) {
    ...
} else if (boolean) {
...
} else {
    ...
}
```

- Structure **atteindre … cas x … cas y …** : embranchement vers un bloc d'instructions énuméré.

```
switch (expression) {
    case constante1 :
        instr11;
        instr12;
        break;

    case constante2 :
        ...
    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 au case suivant sont exécutés.

Il est possible d'imbriquer des switch

- L'opérateur ternaire : (condition) ? valeur-vrai : valeur-faux

Exemple :

```
1.if(niveau == 5) // equivalent à total = (niveau ==5) ? 10 : 5;
2.total = 10;
3.elsetotal = 5;
4.System.out.println((sexe == " H ") ? " Mr ": " Mme ");
```

3.7.3. Les débranchements

break : permet de quitter immédiatement une boucle ou un branchement. Utilisable dans tous les contrôles de flot

continue : s'utilise dans une boucle pour passer directement à l'itération suivante

break et continue peuvent s'exécuter avec des blocs nommés. Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break.

4- Structure d'un programme Java

Un programme Java est constitué d'un ensemble de classes. Aucune partie de code ne peut être écrite en dehors d'une classe. Même la fonction main( ), point d'entréedes applications Java est incluse dans une classe.

Le fichier .java doit obligatoirement porter le nom de la classe (ou si il en a plusieurs,le nom de celle qui est publique).

Une application Java (programme non inclus dans un page Web) doit toujours posséderune classe contenant une méthode appelée main. Cette fonction est celle qui estexécutée en premier et qui appelle toutes les autres (c'est le programme principal enquelque sorte). Plus précisément elle commence ainsi :public static void main (String args[ ])

Donc un programme java est une classe doté de la méthode main

La définition de cette classe **peut être** précédée par une partie importation dans laquelle on fait recours à des services défini dans d'autre bibliothèques.

Exemple :

```
import java.util.*;// partie importation

classe Prog// Prog est le nom de la classe (nom du programme)

{       // début de classe

public static void main (String [] args)

{// début du programme principal

// Les instructions de programme principal

System.out.println("bonjour le prog principale est mis dans la partie main");

//……

}// fin du programme principal

}               // fin de la class
```