

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

INTRODUCTION

Java est un langage de programmation orienté objet développé par Sun Microsystems. Il fut présenté officiellement en 1995 et fut un langage révolutionnaire lors de sa création. Depuis il a beaucoup évolué et est devenu un langage de programmation très puissant permettant de presque tout faire.

Sa force principale réside dans son mode de compilation qui lui permet d'être facilement utilisable sur tous les systèmes d'exploitation. Cette force a été la source d'inspiration pour le slogan du langage: "Write Once, Run Everywhere". ("Écrire le code une fois, l'utiliser partout !").

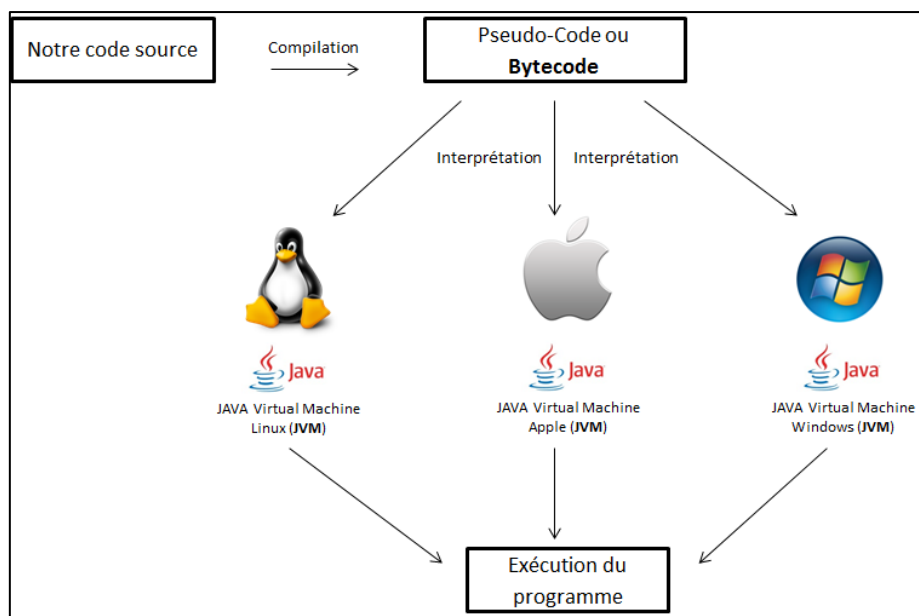


Figure 1: Principe de fonctionnement d'un langage compilé

Aujourd'hui, la technologie Java est à la base de la plupart des applications en réseau et elle est exploitée dans le monde entier pour développer et fournir des applications mobiles et imbriquées, des jeux, du contenu Web et des logiciels d'entreprise.



**FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -**

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

DEROULEMENT DU COURS

I.] LE PREMIER PROGRAMME JAVA

- 1.) Définition – Un programme
- 2.) L'environnement de développement
- 3.) La première application Java

II.] UN PEU DE SYNTAXE JAVA

- 1.) Les identificateurs
- 2.) Les variables
- 3.) Les instructions
- 4.) Attribuer une valeur à une variable
- 5.) Blocs de code
- 6.) Les commentaires
- 7.) Utilisation de fonctions
- 8.) Entrées/sorties en mode console
- 9.) Conventions d'écriture
- 10.) Exemple complet

III.] LES TYPES DE BASE

- 1.) Les 8 types de base
- 2.) Les types entiers
- 3.) Les types réels
- 4.) Le type caractère
- 5.) Le type booléen
- 6.) Les constantes
- 7.) Conversions de types

IV.] OPERATEURS

- 1.) Opérateurs arithmétiques
- 2.) Opérateurs de comparaison

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

- 3.) Opérateurs logiques
- 4.) Opérateurs d'affectation
- 5.) Opérateurs bit à bit

V.] LES STRUCTURES DE CONTROLE

- 1.) Tests conditionnels
- 2.) Boucles
- 3.) Instructions de contrôle des boucles

VI.] LES TABLEAUX

- 1.) Tests conditionnels
- 2.) Boucles
- 3.) Instructions de contrôle des boucles
- 4.) Déclaration d'un tableau
- 5.) Création et initialisation des tableaux
- 6.) Accès aux éléments des tableaux
- 7.) Taille d'un tableau
- 8.) Références de tableaux
- 9.) Les chaînes de caractères

VII.] LES FONCTIONS

- 1.) Introduction
- 2.) Les fonctions prédéfinies
- 3.) Les fonctions personnalisées
- 4.) Passage de paramètres d'entrée par valeur
- 5.) Passage de paramètres d'entrée par référence
- 6.) Paramètre de retour
- 7.) Portée des variables
- 8.) Fonctions récursives

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

I.] LE PREMIER PROGRAMME JAVA

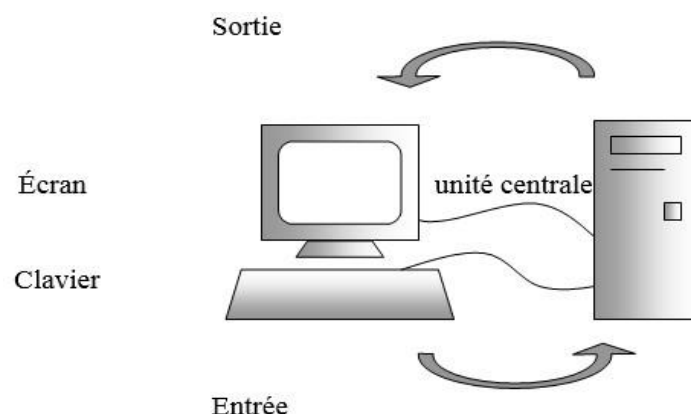
1.) Un programme ?

Un programme est une séquence d'instructions qu'un ordinateur peut exécuter pour effectuer une tâche déterminée. Ceci signifie que:

- l'ordinateur doit "comprendre" les instructions, ce qui nécessite qu'elles soient écrites dans un certain langage de programmation
- l'ordinateur n'effectue rien spontanément, mais uniquement ce que le programmeur lui commande
- l'ordre des instructions est important
- pour effectuer une autre tâche, il faut écrire un autre programme
-

Un programme informatique manipule de l'information stockée dans la mémoire de l'ordinateur.

En général, il acquiert de l'information issue de l'utilisateur, la traite, et restitue une autre information au monde extérieur; les échanges d'informations entre le monde extérieur et l'unité centrale de l'ordinateur se font au travers de périphériques d'entrée (clavier, souris, ...) et de sortie (écran, imprimante, ...).



FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Les langages de programmation ne tolèrent pas l'ambiguïté du langage courant : ils reposent tous sur une **syntaxe**, c'est-à-dire un ensemble de règles qui précisent ce qui est permis. Cette syntaxe définit d'une part les mots de vocabulaire autorisés, et d'autre part les règles de construction d'un programme, telles que les tests, les boucles, les appels de fonctions, ...

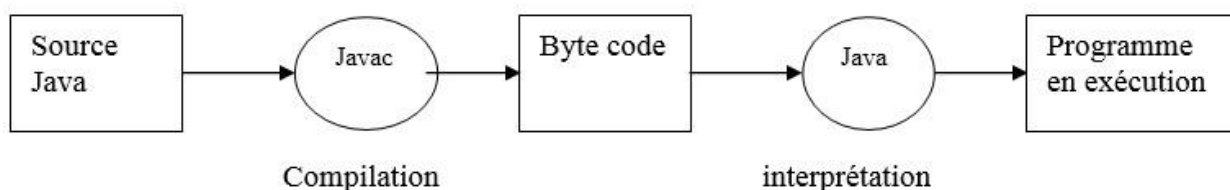
Cependant, qu'un programme soit correct sur le plan de la syntaxe ne suffit pas : il faut encore, pour qu'il atteigne son objectif, qu'il possède une **sémantique** correcte, c'est-à-dire que ce qu'il exécute ait un sens au regard de la tâche souhaitée.

La maîtrise de la syntaxe passe par l'apprentissage du langage ; une sémantique correcte s'obtient en élaborant un algorithme correct, c'est-à-dire une séquence d'instructions qui permettra que le programme fasse ce qu'il doit faire.

2.) L'environnement de développement

Une source Java est un fichier texte contenant des instructions Java. La source peut être créée par un éditeur de texte ou un environnement de développement intégré.

Le bytecode est un ensemble d'instructions codées, et interprétable par un logiciel particulier: la machine virtuelle Java :



Les outils les plus simples sont ceux fournis par SUN, la société qui a promu le langage Java: il s'agit des outils du JDK (Java Development Kit); parmi ceux-ci,

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

`javac` : compilateur Java

`java` : machine virtuelle Java, permettant d'exécuter les instructions (bytecode) fournies par le compilateur Java

Pour compiler, il faut :

- se positionner dans le répertoire contenant le fichier source `cd <répertoire source>`
- utiliser la commande `javac` suivie du nom du fichier source `javac <nom du source>`

Cela crée un fichier pour chaque classe contenue dans le fichier compilé. Ces fichiers ont pour nom la classe correspondante, suivi de l'extension `.class`.

Pour exécuter, il faut :

- Avoir un fichier contenant une classe qui elle-même contient une fonction `main` (voir plus loin) ; □
Taper `java` suivi du nom (sans extension) de la classe contenant la fonction `main()`.
`java <nom de la classe>`

Une autre solution pour développer des programmes Java est d'utiliser un environnement de développement intégré, tel qu'Eclipse ou NetBeans. Nous adopterons l'IDE d'Eclipse pour cette formation.

Organisation du développement

Quel que soit l'outil de développement, il importe d'organiser les fichiers sources des différentes applications que nous allons développer. Pour cela, il faut

- se créer un répertoire personnel consacré au développement Java.
- créer, dans ce répertoire, un sous-répertoire par application (projet).
- veiller à ce que tous les fichiers `.java` et `.class` nécessaires au projet soient effectivement présents dans le répertoire de l'application; (au besoin, les recopier à partir d'un autre répertoire).

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

3.) La première application Java

Voici quelques règles de syntaxe pour démarrer:

- une instruction Java se termine par un ;
- une instruction Java peut être écrite sur plusieurs lignes
- un commentaire sur une ligne commence par //
- un commentaire sur plusieurs lignes commence par /* et se termine par */

Notre première application va afficher « Bonjour! » à l'écran. La voici :

```
public class Hello
{
    /* un programme qui affiche
    "Bonjour" à l'écran          */

    public static void main(String[] args)
    {
        System.out.println("Bonjour!");
    }
}    // fin de la classe Hello
```

Cet exemple nécessite quelques commentaires :

Tout programme Java comporte au moins une classe, d'où la première ligne `public class Hello`. La classe est le composant de base d'une application Java.

- toute application doit posséder un point d'entrée, ici le `main`, qu'on peut appeler de l'extérieur, d'où le mot `public` ; le mot `static` est obligatoire
- les instructions apparaissent au sein d'un bloc, délimité par `{` et `}`



FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

- la ligne qui effectue l'affichage proprement dit est `System.out.println("Bonjour!");` ; Cette instruction consiste à appeler une fonction prédéfinie (un ensemble d'instructions), `System.out.println()` , qui affiche des informations à l'écran, et à lui préciser ce qu'elle doit afficher , en l'occurrence : "Bonjour! "

Saisissons au clavier ce programme à l'aide de l'éditeur de texte, en prenant garde aux majuscules et minuscules!

Pour que ce programme puisse être réutilisé par la suite, il faut le sauver dans un fichier. Le compilateur Java impose que le nom du fichier soit celui de la classe, suivi de l'extension .java.

Donc le fichier qui contiendra notre programme doit s'appeler *Hello.java*

Compilons-le : `javac Hello.java`

Exécutons-le : `java Hello`

Nous obtenons à l'écran l'affichage suivant :

```
Bonjour !
```

II.] UN PEU DE SYNTAXE JAVA

Pour aller plus avant dans la programmation Java, il est nécessaire d'en approfondir la syntaxe.

1.) Les identificateurs

Les langages actuels, tels que Java, permettent d'associer un nom à chaque donnée (variable) ; ce nom correspond à un emplacement spécifique à chaque donnée (une adresse en mémoire) ; le contenu de cet emplacement peut évoluer en fonction du déroulement du programme.

Un des rôles du compilateur est de convertir des noms de variables en des adresses ; la machine virtuelle java manipule effectivement les données par leur emplacement.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Un identificateur est un nom qui identifie de façon unique une variable, une fonction, une classe (nous aborderons les classes plus tard), Dans la plupart des langages, des restrictions s'appliquent à la composition des identificateurs. Voici les restrictions de Java relatives aux identificateurs :

- Tous les identificateurs doivent commencer par une lettre, un trait de soulignement (_) ou un signe dollar (\$)
- Un identificateur peut inclure des chiffres, mais ne doit pas commencer par un chiffre
- Un identificateur ne peut pas contenir d'espace vide (tabulation, espace, nouvelle ligne ou retour chariot)
- Les identificateurs différencient les minuscules et les majuscules
- Les mots-clés de Java ne peuvent pas servir d'identificateurs : ce sont des mots réservés et utilisés par le langage lui-même.

Voici la liste des mots-clés Java, et qui sont réservés pour un usage ultérieur :

abstract	else	int	static
boolean	extends	interface	super
break	false	long	switch
byte	final	native	synchronized
byvalue*	finally	new	this
case	float	null	throw
cast*	for	operator*	throws
catch	future*	outer*	transient
char	generic*	package	true
class	goto*	private	try
const*	if	protected	var*
continue	implements	public	void
default	import	rest*	volatile
do	inner*	return	while
double	instanceof	short	

2.) Les variables

Comment accéder à une donnée de la mémoire (une variable), et comment lui attribuer des valeurs ?

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Pour pouvoir être utilisée, une variable doit être **déclarée**, sous la forme:

```
<type de variable> <identificateur de la variable>;
```

Cette instruction "réserve" un espace mémoire pour la variable, selon son type (voir chapitre suivant). Par exemple

```
int n; /*déclare la variable n à stocker selon le type de données int  
*/ float dX, dY; boolean bFinDeFichier; char carl;
```

Il est possible de déclarer en une seule fois plusieurs variables du même type, en séparant leurs identificateurs par des virgules ; il est cependant préférable, pour la relecture du code, de ne définir qu'une variable par ligne, et d'y adjoindre un commentaire.

De plus, il faut **initialiser** cette variable, c'est-à-dire lui donner une valeur initiale.

```
short nAge;  
nAge=17;
```

Une variable peut être initialisée lors de sa déclaration

```
<type de variable> <identificateur de la variable> = <valeur initiale>;
```

Par exemple

```
double dSalaire = 12345.56;  
float dX, dY = 4.1f, dZ =  
2.2f; boolean bFinDeFichier =  
false; char carl = 'T';
```

Une variable peut être déclarée n'importe où dans un bloc d'instructions (entre { et }).

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Il faut cependant que la déclaration et l'initialisation précèdent la première utilisation de la variable. Pour éviter tout problème, comme référencer une variable qui n'a pas encore été déclarée, il est conseillé de déclarer toutes les variables au début des blocs de code où elles sont utilisées.

Une variable n'est visible que dans le bloc au sein duquel elle est déclarée. (Voir plus loin la notion de portée)

3.) Les instructions

Pour le compilateur, une instruction représente une commande unique. Cependant, toute ligne de code n'est pas nécessairement une instruction : certaines instructions, comme une instruction contenant un if, peuvent se composer de plusieurs lignes de code.

Pour que le compilateur sache où se termine chaque instruction et où commence la suivante, il faut séparer les instructions par des points-virgule.

Puisque les instructions sont toujours séparées par des points-virgule, le compilateur Java ne s'intéresse pas à la longueur et à la disposition de chacune. Par exemple, les deux instructions suivantes sont équivalentes :

```
x = (y + z) / q; //instruction  
1 x = (y + z  
  ) / q;      //instruction 2
```

La deuxième instruction englobe des caractères espace vide (les caractères espace vide sont l'espace, les tabulations horizontale et verticale, le saut de page et la nouvelle ligne). Bien que le compilateur Java ignore tous les caractères espace vide intégrés aux instructions, il est déconseillé d'y avoir recours pour ne pas rendre le code plus difficile à lire.

4.) Attribuer une valeur à une variable

En Java, pour donner une valeur à une variable, on effectue une affectation, c'est-à-dire une instruction de la forme

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

`<identificateur de variable> = <expression> ;`

Attention, le signe = est l'opérateur d'affectation; il n'a pas la même signification qu'en mathématiques!

L'affectation est un exemple d'instruction. Dans cette affectation, <expression> désigne toute chose qui peut être évaluée (à laquelle Java peut donner une valeur) : une constante, une variable, ...

Par exemple :

```
nAge = 4;
```

signifie que la variable `nAge` prend la valeur 4, ou que la valeur 4 est rangée à l'emplacement de la variable `nAge`, quelle que soit la valeur qui s'y trouvait précédemment. Ou bien :

```
dInteret = dTaux * dCapital;
```

signifie que la variable `dIntérêt` prend comme valeur le produit de la valeur de `dTaux` par la valeur de `dCapital`.

5.) Blocs de code

Un bloc de code est un groupe d'instructions qui se comportent comme une unité. Java délimite les blocs de code par des accolades (`{et}`). Les définitions des classes, les instructions de boucle, les instructions conditionnelles, les instructions d'exception et les corps des fonctions sont des exemples de blocs de code. Dans l'exemple de code suivant, il y a trois blocs de code : la fonction `lireI()`, le bloc `try` et le bloc `catch`.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
public static int lireI() // Lire un entier
{ int n = 0; try {      n =
    Integer.parseInt(lireStr());
  } catch (NumberFormatException
    e)
  {
    System.out.println("Format incorrect");
    System.exit(0);
  }
  return n;
}
```

Le code ci-dessus illustre le concept de blocs imbriqués : les blocs try et catch sont imbriqués dans le bloc principal lireI().

6.) Les commentaires

Les commentaires permettent au programmeur d'ajouter des remarques concernant le code. Dans Java, il y a trois styles de commentaires. Le premier commence par /* et se termine par */ ; il permet d'écrire des commentaires sur plusieurs lignes.

Le code suivant montre l'utilisation de ce style :

```
x = y + z; /* Ceci est un
commentaire.*/ z = q / p; /*Ce
commentaire
s'étend sur deux lignes*/
```

Quand le compilateur Java rencontre /*, il ignore tout ce qui suit jusqu'à ce qu'il rencontre */.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Le deuxième style de commentaire ressemble au premier mais commence par `/**` et se termine par `*/`. La différence est que ce style est utilisé par l'outil javadoc du JDK pour générer automatiquement une documentation à partir du code source.

Le commentaire de troisième style commence par `//` et ne peut être écrit que sur une ligne. Voici un exemple :

```
x = y + z; //Ce commentaire doit être écrit sur 1 ligne
```

L'imbrication de commentaires n'est possible que lorsque des commentaires du troisième style sont imbriqués dans un des deux autres styles. L'imbrication des commentaires des deux premiers styles n'est pas autorisée.

Voici un commentaire imbriqué **invalide** :

```
/*Début du commentaire  
/*  
Le commentaire se termine ici  
*/  
Ceci ne se trouve pas dans le commentaire et provoquera une erreur  
du compilateur */
```

En effet, le compilateur ignore tout ce qui se trouve entre `/*` et `*/` ; aussi, quand il rencontre le premier `*/` il pense que le commentaire se termine. En conséquence, la dernière ligne du code ne se trouve pas dans le commentaire.

Voici un exemple de commentaire imbriqué valide :

```
/*Début du commentaire  
//Ceci est correct  
//Ceci aussi  
Fin du commentaire.  
*/
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

7.) Utilisation de fonctions

Java permet de regrouper plusieurs instructions et d'affecter un nom à ce regroupement: il s'agit d'une **fonction**. Au lieu d'écrire toutes les instructions dans notre source, on écrit le nom de la fonction: on dit qu'on appelle la fonction. De plus, le déroulement des instructions de cette fonction peut être modifié en donnant une ou des informations supplémentaires à la fonction: il s'agit de paramètres. Dans une première étape, nous nous bornerons à utiliser des fonctions; ultérieurement, nous écrirons nos propres fonctions. (Voir le chapitre consacré aux fonctions).

Le langage Java permet d'utiliser un grand nombre de fonctions prédéfinies (déjà écrites) : elles sont regroupées en paquets (bibliothèques), qui eux-mêmes peuvent être structurés. Dans certaines situations, nous serons amenés à préciser le paquetage auquel appartient la fonction ; ces notions de paquetage seront précisées ultérieurement. Dans l'immédiat, l'expression `System.out.println()` peut être considérée comme un tout.

8.) Entrées/sorties en mode console

Le langage Java a été conçu pour des applications en mode graphique (applets, ...). Les entrées au clavier d'une application en mode console (non graphique) sont d'un abord délicat pour les débutants en programmation.

Pour éviter cet écueil et sans trop développer le sujet pour le moment, nous allons faire appel à une instance de la classe `Scanner` qui regroupe un grand nombre de fonctions. Ces fonctions nous permettront de charger dans des variables des valeurs saisies au clavier. Pour les types de base, ces fonctions sont les suivantes :

<code>nextByte()</code>	lecture d'un byte	// valeur retournée : byte
<code>nextDouble()</code>	lecture d'un double	// valeur retournée : double
<code>nextFloat()</code>	lecture d'un float	// valeur retournée : float
<code>nextInt()</code>	lecture d'un int	// valeur retournée : int
<code>nextLong()</code>	lecture d'un long	// valeur retournée : long
<code>nextShort()</code>	lecture d'un short	// valeur retournée : short

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Pour les sorties écran d'une application console, nous utiliserons

`System.out.print()` écriture

`System.out.println()` écriture et retour à la ligne

Voici un exemple d'utilisation de ces fonctions

```
Import java.util.Scanner;

public class LireEcrire
{
    public static void main( String args[] )
    {
        int n;
        Scanner sc = new Scanner(System.in);

        System.out.println("entrez un entier");
        n = sc.nextInt();
        System.out.println(n);
        byte nOctet;
        System.out.println("entrez un byte");
        nOctet = sc.nextByte();
        sc.nextLine(); // prend en compte le retour chariot
        System.out.println(nOctet);
        System.out.println("c'est fini. Tapez <ret>");
        sc.nextLine();
    }
}
```

Dans certaines situations, il est nécessaire de manipuler des variables dont la valeur doit rester fixe durant toute l'exécution du programme. C'est le cas, par exemple, de variables qui représentent les dimensions de l'écran : ces dimensions n'évoluent pas au cours du déroulement de l'application.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

A cet effet, Java prévoit un mot-clé, *final*, qui permet de préciser au compilateur que la valeur de cette variable ne peut jamais changer. Evidemment, la valeur de cette variable constante ne peut être attribuée qu'au moment de la déclaration de la variable ! Par exemple :

```
final int LARGEUR_ECRAN = 640;
```

De plus si cette variable à la valeur fixe doit être accessible depuis plusieurs fonctions de la classe , il faut ajouter le mot-clé *static* :

```
final static int LARGEUR_ECRAN = 640;
```

Ce mot-clé *static* recouvre une notion bien plus vaste, qui sera abordée par la suite.

9.) Conventions d'écriture

Bien que ne faisant pas partie de la syntaxe Java, certaines conventions d'écriture sont couramment admises ; en particulier, elles facilitent la lecture du code. Le tableau suivant donne la liste des conventions couramment retenues pour désigner des identificateurs selon leur catégorie ; les identificateurs peuvent être simples ou composés (ils comprennent plusieurs mots) :

Catégorie d'identificateur	Convention	Exemples
Nom de classe	nom ou nom+adjectif ; chaque mot commence par une majuscule	Element, ElementMobile
Nom de fonction	verbe ou verbe+complément ; chaque mot, sauf le premier, commence par une majuscule	obtenirPosition, définirHauteur
Nom de variable	nom ; chaque mot, sauf le premier, commence par une majuscule	position, vitesseDeplacement
Noms des constantes	nom ; toutes les lettres sont en majuscules et les mots sont séparés par un trait de soulignement	MAX_HEIGHT, TAILLE_MINI

Le tableau suivant donne la liste des conventions couramment retenues pour désigner les identificateurs de variables selon leur type (les identificateurs de variables sont préfixés) :

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Préfixe	Type	Exemple
b	booléen (vrai/faux)	(bOK)
n	nombre entier	(nAge)
l	nombre entier long	(lAge)
d (ou f)	nombre en virgule flottante	(dSalaire)
i	itérateur (dans une boucle)	(iCpt)
str	chaîne de caractères	(strNom)

Au niveau de l'écriture du code, voici les conventions retenues :

- Une seule instruction par ligne.
- Les délimiteurs d'un bloc { et } doivent se trouver sur des lignes différentes et être alignés sur la première colonne de sa déclaration.
- A l'intérieur d'un bloc { } les instructions sont indentées (décalées) par un caractère *tabulation*.
- A l'intérieur d'un bloc { } la partie *déclaration des variables* et la partie *instructions* sont séparées par une ligne vide.

Ces conventions élémentaires, ainsi que d'autres, figurent dans le document : « Conventions d'écriture programmes Java ».

10.) Exemple complet

Ecrivons un programme qui calcule l'aire d'un cercle et le volume d'une sphère à partir de la valeur du rayon saisie au clavier. Voici l'analyse de ce problème :

Constante

PI = 3.1416

Variables

entrées

dR: double rayon

internes

dS: double surface

dV: double volume

Traitement

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

début

```
saisir dR au clavier;  
dS <- PI*dR*dR;  
dV <- 4*PI*dR*dR*dR/3;  
afficher("la surface vaut: ");  
afficher(dS);  
afficher("le volume vaut: ");  
afficher(dV);
```

fin

Le programme correspondant en Java est le suivant

```
Import java.util.Scanner;  
public class Expl01  
{    final static double PI=3.1416;  
  
    public static void main(String[] args)  
    {  
        double dR; // Rayon          double  
dS; // Surface          double dV; // Volume  
sphère          Scanner sc = new  
Scanner(System.in);  
  
        System.out.println("Entrer le rayon: ");  
  
        dR = sc.nextDouble();  sc.nextLine(); dS = PI  
        * dR * dR;  
        dV = 4.0 * PI * dR * dR * dR / 3.0;  
        System.out.print("La surface vaut: ");  
        System.out.println(dS);  
        System.out.print("Le volume vaut: ");  
        System.out.println(dV);  
  
    }  
}
```

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

III.] LES TYPES DE BASE

1.) Les 8 types de base

Les types de données représentent des catégories spécifiques de valeurs pouvant être stockées en mémoire et interprétées d'une façon spécifique par le compilateur.

Java intègre des types de données de base (ou primitifs) qui couvrent les trois catégories numérique, booléen et caractère. La taille des types de base est définie dans le langage et elle est donc indépendante de la machine ou du système d'exploitation.

Les 8 types de base en Java sont les suivants :

Type	Occupation mémoire (bits)	Utilisation	Valeurs mini, maxi
byte	8	Entier signé	$-2^7, 2^7-1$
short	16	Entier signé	$-2^{15}, 2^{15}-1$
int	32	Entier signé	$-2^{31}, 2^{31}-1$
long	64	Entier signé	$-2^{63}, 2^{63}-1$
float	32	Réel	7 chiffres significatifs
double	64	Réel	15 chiffres significatifs
char	16	Caractère Unicode	
boolean	1	Booléen (vrai, faux)	True, false

2.) Les types entiers

Pour représenter les types entiers, Java utilise:

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



FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Tous les types numériques sont des types signés. Il n'existe pas de type numérique non signé. Des valeurs spéciales sont définies pour représenter les valeurs extrêmes des nombres entiers :

`java.lang.Byte.MIN_VALUE` et `java.lang.Byte.MAX_VALUE`

`java.lang.Short.MIN_VALUE` et `java.lang.Short.MAX_VALUE`

`java.lang.Int.MIN_VALUE` et `java.lang.Int.MAX_VALUE`

`java.lang.Long.MIN_VALUE` et `java.lang.Long.MAX_VALUE`

3.) Les types réels

Pour représenter les types réels, Java utilise:

float : 32 bits 3.40282347E+28 à 1.40239846E-45

double : 64 bits 1.79769313486231570E+308 à 4.9406545841246544E-324

Des valeurs spéciales sont également définies pour représenter les valeurs extrêmes des nombres réels.

4.) Le type caractère

Pour représenter le type caractère, Java utilise:

Char : 16 bits 0 à 65535

Le jeu de caractères Java est le jeu Unicode (16 bits non signé) permettant ainsi de coder les alphabets du monde entier. Les 128 premiers caractères Unicode sont les mêmes que ceux du code ASCII.

5.) Le type booléen

Pour représenter le type booléen, Java utilise:

boolean : 1 bit true ou false

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Le type `boolean` ne peut recevoir que 2 valeurs : `true` ou `false` et n'est pas compatible avec un type numérique.

6.) Constantes

Une constante représente une valeur qui ne change jamais. Il importe de distinguer un identificateur qui représente une valeur et une constante qui est une valeur. Par exemple, le nombre 17 est une constante ; l'identificateur `nAge` représente un nombre qui peut être égal à 17.

Dans Java, une constante peut être un nombre (entier ou réel), un booléen, un caractère ou une chaîne de caractères.

Constantes entières

Les constantes entières peuvent prendre trois formats : décimal (base 10), hexadécimal (base 16) et octal (base 8). Les constantes décimales sont écrites comme des nombres ordinaires; les constantes hexadécimales commencent toujours par `0X` ('zéro'X) et les constantes octales commencent par `0` ('zéro'). Par exemple, le nombre décimal 10 s'écrit `0xA` ou `0XA` en format hexadécimal et `012` en format octal.

Une constante entière peut être stockée dans les types de données `byte`, `short`, `int` ou `long`. Par défaut, Java stocke les constantes entières dans le type de données `int`, limité à 32 bits.

Pour stocker une constante entière dans le type de données `long`, qui peut recevoir des valeurs de 64 bits, il faut ajouter le caractère `L` ou `l` à la fin de la constante. Par exemple, la constante `9999L` est stockée dans le type `long`. Les lignes de code suivantes utilisent des constantes entières :

```
int nx = 12345;    //12345 est une constante
int ny = nx * 4;   //4 est une constante
```

Dans la première ligne, la constante 12345 est stockée directement dans la variable `nx` de type `int`. Dans la deuxième ligne, la constante 4 sert à calculer une valeur qui est ensuite stockée dans la variable `ny` de type `int`.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Constantes réelles

Une constante réelle est un nombre contenant un point décimal et/ou un exposant. Une constante réelle suit une notation standard ou scientifique:

123.456	notation standard
1.23456e+2	notation scientifique.

Les constantes réelles sont du type double sur 64 bits (type par défaut) ou du type float sur 32 bits. Pour stocker une constante réelle dans le type float, il faut ajouter la lettre f ou F à la fin du nombre.

Constantes booléennes

Une constante booléenne représente deux états possibles, vrai et faux, par les mots-clés `true` (vrai) et `false` (faux). Les constantes booléennes sont stockées dans le type de données booléen.

Constantes caractère

Une constante caractère représente un caractère Unicode unique. Les constants caractères sont toujours mis entre quotes (' '); par exemple, 'A' et '9' sont des constants caractères. Java utilise le type `char` pour stocker des caractères uniques.

Le jeu de caractères Unicode, sur 16 bits, remplace le jeu ASCII de 8 bits (les 128 premiers caractères Unicode correspondent au jeu ASCII). Il peut définir jusqu'à 65536 valeurs, ce qui permet d'inclure les symboles et les caractères d'autres langues.

Les constantes caractères peuvent posséder une représentation

- normale : 'A'
- octale : 0101
- hexadécimale : 0x41
- Unicode : '\u0041' (4 chiffres hexa maximum)
-

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Certains caractères spéciaux utilisent un backslash (\) comme caractère d'échappement: en particulier

'\r'	//carriage return	retour chariot
'\n'	//line feed	nouvelle ligne
'\t'	//tab	tabulation horizontale
'\"'	//quote	
'\\'	//backslash	
'\'	//slash	continuation

Constantes chaînes de caractères

Une constante chaîne de caractères représente une suite de caractères. Dans Java, les chaînes de caractères sont toujours entre guillemets doubles. Java gère les chaînes avec les classes String (voir plus loin) Exemple :

```
"Apiai\n"
```

7.) Conversions de types

Dans certains cas, il est nécessaire de convertir un type de variable en un autre type. Par exemple, il peut être nécessaire de transmettre une variable de type int à une fonction qui n'accepte que des variables de type float. Convertir le type d'une variable s'appelle transtypage, ou *cast* en anglais. Pour transtyper une variable, il suffit de faire précéder l'identificateur de la variable du type voulu entre parenthèses. L'exemple suivant montre comment la variable nx, de type int, peut être transtypée en type float :

```
int nx = 3;  
float df = (float) nx;
```


FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Les transtypes sont des opérations délicates, car il y a des risques de perte d'informations. Par exemple, lors d'un transtypage d'une variable de type long de 64 bits en une variable de type int de 32 bits, le compilateur ignore les 32 bits de poids fort de la variable de type long. Si la valeur de la variable de type long dépasse 32 bits au moment du transtypage, le transtypage attribue une valeur inexacte à la variable de type int.

En règle générale, le type cible du transtypage doit avoir une taille au moins égale au type initial. Le tableau suivant montre les transtypes qui ne risquent pas de provoquer des pertes d'informations :

Type initial	Transtypage
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, <u>float</u> , double
long	float, double
<u>float</u>	double

Transtypage implicite

Dans certains cas, un transtypage peut être réalisé implicitement par le compilateur. Voici un exemple :

```
int nx = 0x41; if (nx > 'a')
{
    ...
}
```

Dans ce cas, la valeur 'a' est convertie en nombre entier (valeur ASCII de la lettre a) avant d'être comparée au nombre 3.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

IV.] LES OPERATEURS

Les opérateurs sont des symboles spéciaux qui exécutent une fonction particulière sur des opérands. Il y a cinq types généraux d'opérateurs :

- opérateurs arithmétiques
- opérateurs de comparaison
- opérateurs logiques
- opérateurs d'affectation
- opérateurs au niveau bits.

On distingue également les opérateurs unaires, utilisant un seul opérande, et les opérateurs binaires, utilisant deux opérands.

1.) Opérateurs arithmétiques

Ces opérateurs peuvent être utilisés avec tout type numérique : byte, short, int, long, float, double.

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
+	binaire	addition	$x + y$, $5.9 + 8.6$
-	binaire	soustraction	$x - y$, $7 - 3$
*	binaire	multiplication	$x * y$, $8 * 5$
/	binaire	division	x/y , $7/2$
%	binaire	modulo	$34\%7$, $x \% y$
-	unaire	complémentation	$-x$

Les opérands peuvent être des constantes, des variables ou des expressions. Si les 2 opérands d'un opérateur binaire sont de types différents, il y aura conversion dans le type le plus « grand » avant que l'opération soit effectuée. Attention, l'opérateur / entre deux entiers correspond à la division entière et donne un résultat entier.

Voici trois exemples :

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
public class CalculEntier
{
    public static void main (String args[])
    {
        int nX = 17, nY = 5;    System.out.println(nX);
        System.out.println(nY);
        System.out.println(nX + nY);
        System.out.println(nX - nY);
        System.out.println(nX * nY);
        System.out.println(nX / nY);
        System.out.println(nX % nY);
    }
}
```

```
public class CalculFloat
{
    public static void main (String args[])
    {
        float dX = 23.5f, dY = 7.3f;
        System.out.println(dX);
        System.out.println(dY);
        System.out.println(dX + dY);
        System.out.println(dX - dY);
        System.out.println(dX * dY);
        System.out.println(dX / dY);
        System.out.println(dX % dY);
    }
}
```

```
public class Complement
{
    public static void main (String args[])
    {
        int nX = 8;
        System.out.println(nX);
        int nY = -nX;
        System.out.println(nY);
    }
}
```

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Les opérateurs ++ (et --) permettent d'incrémenter (resp. décrémenter) la valeur d'une variable.

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
++ (préfixe)	unaire	incrémenter	++ x
++(postfixe)	unaire	incrémenter	x ++
--(préfixe)	unaire	décrémenter	-- x
--(postfixe)	unaire	décrémenter	x --

Dans le cas d'expressions simples, les notations préfixe et postfixe donnent le même résultat. Par contre, dans le cas d'affectations comme

```
nZ = ++nX;
```

La valeur de nX est attribuée à nZ après incrémenter, alors que dans l'affectation

```
nZ = nX++;
```

La valeur de nX est incrémentée après l'affectation de nX à nZ. Voici un exemple utilisant les opérateurs d'incrémenter :

```
public class IncrDecr
{
    public static void main (String args[])
    {
        int nX = 8, nY = 13;
        int nZ = 0;
        System.out.println(nX);
        System.out.println(nY);
        System.out.println(++nX);
        System.out.println(nY++);
        System.out.println(nX);
        System.out.println(nY);
    }
}
```

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
nZ = ++nX;  
System.out.println(nZ);  
nZ = nY++;  
System.out.println(nZ);  
}  
}
```

2.) Opérateurs de comparaison

Ces opérateurs servent à comparer des valeurs ; ils retournent une valeur booléenne.

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
<	binaire	Inférieur à	x < 5
>	binaire	Supérieur à	x > 5
<=	binaire	Inférieur ou égal à	x <= 5
>=	binaire	Supérieur ou égal à	x >= 5
==	binaire	Egal à	x == 6
!=	binaire	Différent de	x != 5

Voici un exemple :

```
public class Comparaison  
{  
    public static void main (String args[])  
    {  
        int nX = 7, nY = 11, nZ = 11;  
        System.out.println(nX);  
        System.out.println(nY);  
        System.out.println(nX < nY);  
        System.out.println(nX > nZ);  
        System.out.println(nY <= nZ);  
        System.out.println(nX >= nY);  
        System.out.println(nY == nZ);  
        System.out.println(nX != nZ);  
    }  
}
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

3.) Opérateurs logiques

Ces opérateurs n'acceptent que des opérandes de type *boolean* (valant *true* ou *false*). Ils permettent de regrouper des expressions booléennes pour déterminer des conditions.

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
&&	binaire	ET logique	<code>a>b && b>c</code>
	binaire	OU logique	<code>a>b b>c</code>
!	unaire	NON logique	<code>!(a<b)</code>

Les opérateurs && et || évaluent toujours le premier opérande et, si cela suffit à déterminer la valeur de la totalité de l'expression, ils n'évaluent pas le deuxième opérande.

Ainsi, dans les lignes suivantes :

```
if (!bHautePression && (temperature1 > temperature2))  
{  
    ...  
}  
boolean1 = (x < y) || (a > b);  
boolean2 = (10 > 5) && (5 > 1);
```

La première instruction évalue d'abord `!bHautePression` ; si le résultat a la valeur *false*, elle n'évalue pas le deuxième opérande (`temperature1 > temperature2`), puisque le premier opérande ayant la valeur *false*, la totalité de l'expression a forcément la valeur *false*. La deuxième instruction agit de même : `boolean1` n'a la valeur *true* que si `x` est inférieur à `y` (la valeur du deuxième opérande n'est jamais déterminée). Dans la troisième instruction, le compilateur calcule les valeurs des deux opérandes avant d'attribuer le résultat à `boolean2`.

4.) Opérateurs d'affectation

Le tableau suivant donne la liste des opérateurs d'affectation en Java :

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
=	binaire	Affectation	x = 2
+=	binaire	Ajout et affectation	x += 2
-=	binaire	Soustraction et affectation	x -= 3
*=	binaire	Multiplication et affectation	x *= 4
/=	binaire	Division et affectation	x /= 5
&=	binaire	AND avec affectation	z &= 0x22
=	binaire	OR avec affectation	z = 0x31
^=	binaire	XOR avec affectation	z ^= 0x55

Le premier opérateur effectue une simple affectation. Les autres opérateurs d'affectation commencent par une opération, puis stockent le résultat de l'opération dans l'opérande situé sur le côté gauche de l'expression. Voici quelques exemples :

```
int nY = 2;
nY *= 2; //identique à (nY = nY * 2)
boolean b1 = true, b2 = false;
b1 &= b2; //identique à (b1 = b1 & b2)
```

5.) Opérateurs bit à bit

Ces opérateurs n'acceptent que des opérandes de type *entiers*

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
&	binaire	ET binaire (bit à bit)	x & y
	binaire	OU binaire (bit à bit)	x y
^	binaire	XOR binaire (bit à bit)	x ^ y
~	unaire	NON binaire (bit à bit)	~ x

Voici deux exemples d'utilisation des opérateurs bit à bit :

```
public class Binaire
{
```

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
public static void main (String args[])
{
    int nX = 5, nY = 6;
    System.out.println(nX);
    System.out.println(nY);
    System.out.println(nX & nY);
    System.out.println(nX | nY);
    System.out.println(nX ^ nY);
}
```

```
public class ComplementBinaire
{
    public static void main (String args[])
    {
        int nX = 8;
        System.out.println(nX);
        int nY = ~nX;
        System.out.println(nY);
    }
}
```

Java propose également des opérateurs de décalage de bits

Opérateur	Unaire, binaire	Opération effectuée	Exemples d'expression
<<	binaire	Décalage à gauche signé	x << 2
>>	binaire	Décalage à droite signé	x >> 1
>>>	binaire	Décalage à droite par ajout de zéros (non signé)	x >>> 3

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Par exemple:

```
public class Shift
{
    public static void main (String args[])
    {
        int nX = 7;
        System.out.println(nX);
        System.out.println(nX >> 2);
        System.out.println(nX << 1);
        System.out.println(nX >>> 1);
    }
}
```

Les opérateurs de décalage avec affectation existent également.

6.) Opérateur ternaire :

Java possède un opérateur ternaire, l'opérateur ? dont voici la syntaxe :

<expression1> ? <expression2> : <expression3> ;

expression1 est évaluée en premier. Si sa valeur est true, *expression2* est évaluée, sinon *expression3* est évaluée. Voici une illustration :

```
int nX = 3, nY = 4, nMax;
nMax = (nX > nY) ? nX : nY; //ceci revient à dire nMax=nY;
```

Dans ce code, *nMax* reçoit la valeur de *nX* ou de *nY*, selon que *nX* est supérieur à *nY* ou non. Attention de ne pas confondre cet opérateur avec une instruction conditionnelle !

7.) Précédence des opérateurs

Le tableau ci-dessous résume l'ordre de précedence des opérateurs en Java, du plus prioritaire au moins prioritaire.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Opérateurs postfixes	[] . (params) <i>expr</i> ++ <i>expr</i> --
Opérateurs unaires	++ <i>expr</i> -- <i>expr</i> + <i>expr</i> - <i>expr</i> ~ !
Creation ou cast	new (type) <i>expr</i>
Multiplication	* / %
Addition	+ -
Décalage	<< >> >>>
Comparaison	< > <= >= instanceof
Egalité	== !=
ET bit à bit	&
OU exclusif bit à bit	^
OU bit à bit	
ET logique	&&
OU logique	
Condition ternaire	? :
Affectation	= += -= *= /= %= &= ^= = <<= >>= >>>=

La précedence des opérateurs peut être modifiée en entourant des expressions avec des parenthèses : le compilateur évalue d'abord l'expression entre parenthèses.

Par exemple :

```
public class Precedence
{
    public static void main (String args[])
    {
        int nX = 2;
        int nY = 3;
        int nZ = 4;
        int nT = 6;
        System.out.println(nX+nY*nZ-nT/2);
        System.out.println((nX+nY)*nZ-nT/2);
        System.out.println(((nX+nY)*nZ-nT)/2);
    }
}
```

produit l'affichage

```
11
17
7
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

V.] LES STRUCTURES DE CONTROLE

Dans un programme, la structure de contrôle du déroulement la plus simple est la séquence, dans laquelle chaque instruction est exécutée une fois.

Avec les instructions conditionnelles, il est possible d'effectuer ou non une ou plusieurs instructions en fonction d'une condition.

D'autre part, il est souvent nécessaire d'exécuter une ou plusieurs instructions plusieurs fois jusqu'à ce qu'une condition soit satisfaite : ceci conduit aux structures de boucles.

1.) Tests conditionnels

En Java, il y a deux structures conditionnelles : l'instruction if-else et l'instruction switch.

L'instruction if-else

Voici la syntaxe de l'instruction if-else :

```
if (condition1)
{
    //bloc de Code 1
}
else if (condition2)
{
    //bloc de Code 2
}
else
{
    //bloc de Code 3
}
```

L'instruction if-else est généralement constituée de plusieurs blocs. Quand l'instruction if-else s'exécute, le bloc dont la condition a la valeur true est exécuté. Les blocs if-else et le bloc else sont facultatifs. L'instruction if-else n'est pas limitée à trois blocs comme ci-dessus, elle peut contenir autant de bloc if-else que nécessaire.

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Les exemples suivants montrent l'utilisation de l'instruction if-else :

```
if ( nX % 2 == 0)
{
    System.out.println("nX est pair");
}
else
{
    System.out.println("nX est impair");
}
```

```
{
    System.out.println("dX est égal à dY");
}
else if (dX < dY)
{
    System.out.println("dX est inférieur à dY");
}
else
{
    System.out.println("dX est supérieur à dY");
}
```

L'instruction switch

Voici la syntaxe générale de l'instruction switch :

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -


(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
switch (expression)
{
case valeur1:
    ...
    //blocCode1;
    break;
case valeur2:
    ...
    //blocCode2;
    break;
case...
    ...
default :
    //blocCode3;
}
```

L'instruction *switch* appelle certains commentaires :

- Les blocs de code n'ont pas besoin d'être mis entre accolades.
- Le bloc de code *default* correspond au bloc *else* d'une instruction *if-else*
- Les blocs de code sont exécutés selon la valeur d'une variable ou d'une expression, pas d'une condition
- La valeur de l'expression doit être de type nombre entier (ou d'un type qui peut être transtypé en int sans risque, comme *char*). 
- Les valeurs *case* doivent être des expressions constantes du même type que l'expression
- Le mot-clé *break* est nécessaire dans la plupart des cas. Il est nécessaire pour terminer l'exécution de l'instruction *switch* une fois qu'un code de bloc s'exécute. Si, par exemple, il n'est pas mis après *blocCode1* et si *blocCode1* est exécuté, alors *blocCode2* s'exécute immédiatement après *blocCode1* (un effet secondaire parfois utile mais, dans la plupart des cas, indésirable).

Si un bloc de code doit être exécuté quand l'expression a une valeur parmi plusieurs, il faut énumérer les valeurs, chacune étant précédée d'un mot-clé *case* et suivie par un point-virgule

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Voici un exemple (c est de type char) :

```
switch (c)
{
    case '1': case '3': case '5': case '7': case '9':
        System.out.println("c est un chiffre impair");
        break;
    case '0': case '2': case '4': case '6': case '8':
        System.out.println("c est un chiffre pair");
        break;
    default :
        System.out.println("c n'est pas un chiffre");
}
```

L'instruction *switch* évalue *c* et passe directement à l'instruction *case* dont la valeur est égale à *c*. Si aucune des valeurs *case* n'est égale à *c*, la section *default* est exécutée. Il est possible d'utiliser plusieurs valeurs pour chaque bloc.

2.) Boucles

Une boucle est une structure permettant de répéter un ensemble d'instructions (un bloc d'instructions).

En Java, il y a trois façons de créer des boucles : boucles *while*, *do* et *for*.

La boucle *while*

La boucle *while* est utilisée pour créer un bloc de code qui sera exécuté tant qu'une condition particulière est satisfaite. Voici la syntaxe générale de la boucle *while* :

```
<expr initialisation>
while (<expr condition>)
{
    ...
    // code à exécuter dans la boucle
}
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

En général, la structure de boucle est précédée d'une ou plusieurs expressions d'initialisation ; elles permettent de fixer la valeur de certaines variables qui conditionneront le déroulement de la boucle.

La boucle commence par tester la condition. Si la condition possède la valeur true, la boucle exécute la totalité du bloc. Ensuite, elle teste la condition une nouvelle fois et répète ce processus jusqu'à ce que la condition prenne la valeur false. A ce moment, la boucle achève son exécution.

Il est impératif de vérifier que la condition prend la valeur false au moins une fois pour que la boucle se termine ; sinon, elle s'exécute indéfiniment.

En voici un exemple simple :

```
int nx = 0;
//imprime "Cafe" 10 fois
while (nx < 10)
{
    System.out.println("Cafe");
    nx++;
}
```

La variable nx est initialisée à 0. Quand la boucle commence, elle regarde si la valeur de nx est inférieure à 10. Comme c'est le cas, le corps de la boucle est exécuté. Le mot "Café" est affiché à l'écran, puis la valeur de x est incrémentée. La boucle continue jusqu'à ce que la valeur de nx soit supérieure ou égale à 10.

Il est également possible d'arrêter l'exécution d'une boucle avec les instructions return, continue ou break. L'instruction return est illustrée dans le chapitre consacré aux fonctions; les instructions break et continue sont présentées dans la section suivante.

Néanmoins, il est préférable d'explicitement les conditions d'une boucle, et de limiter l'usage des break et continue, qui trahissent une analyse souvent superficielle.

La boucle do

La boucle do ressemble à la boucle while, sauf qu'elle évalue la condition après les instructions et non avant.

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
<expr initialisation>
do
{
    ...
    //code à exécuter dans la boucle
}
while (<expr condition>);
```

La différence principale entre les deux boucles est que, à la différence de la boucle while, la boucle do s'exécute toujours au moins une fois.

☞ Notez-le ; à la suite de l'expression de condition !

Le code suivant montre la boucle while précédente convertie en boucle do :

```
int nx = 0;
do
{
    System.out.println("Cafe");
    nx++;
}
while (nx < 10);
```

La boucle for

La boucle for est la plus puissante des constructions de boucles. Elle permet entre autres d'exécuter des boucles dont le nombre d'itérations est connu d'avance. Voici la syntaxe générale d'une boucle for :

```
for (<expr initialisation>; <expr condition>; <expr opération>)
{
    ...
    // code à exécuter dans la boucle
}
```


FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

La mise en place d'une boucle `for` nécessite trois parties :

- une expression d'initialisation : elle permet d'initialiser la variable l'indice de la boucle. Elle est la première instruction exécutée juste avant l'entrée dans la boucle
- une expression de condition : elle définit la condition à respecter pour continuer à exécuter la boucle. Elle est examinée avant chaque tour de boucle, y compris au premier tour de boucle
- une expression d'opération : cette expression met généralement à jour la variable de la boucle initialisée par la première expression ; en augmentant ou diminuant la valeur de la variable testée, elle modifie la valeur de l'expression ci-dessus. Cette instruction est exécutée à la fin de chaque tour de boucle

Voici la boucle `for` équivalent à notre boucle `while` initiale :

```
int nx = 0;
for (nx = 0; nx < 10; nx++)
{
    System.out.println("Cafe");
}
```

La boucle `for` est plus souple que les autres boucles ; elle peut être construite de façon à diminuer le nombre de lignes de code et à en améliorer l'efficacité. Pour démontrer la souplesse de la boucle `for`, examinons le code suivant :

```
int nx = 1, nz = 0;
while (nx <= 20)
{
    nz += nx;
    nx++;
}
```

Cette boucle additionne simplement les nombres de 1 à 20 (inclus). Voici la boucle `for` équivalente :

```
int nx = 0;
int nz = 0;
for (nx=1; nx <= 20; nx++)
{
    nz += nx;
}
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Pour réduire (artificiellement) de moitié le nombre de fois où la boucle s'exécute, nous pouvons utiliser la boucle suivante :

```
int nx = 0;
int ny = 0;
int nz = 0;
for (nx = 1, ny = 20, nz = 0; nx <= 10 && ny > 10; nx++, ny--)
{
    nz += nx+ny;
}
```

Pour mieux comprendre le fonctionnement de cette boucle, découpons-la en quatre sections principales :

1. L'expression d'initialisation : nx=1, ny=20, nz=0
2. La condition de la boucle : nx<=10 && ny>10
3. L'expression "d'opération" : nx++, ny--
4. Le corps principal : nz+= nx + ny

Boucles imbriquées

Les boucles peuvent être imbriquées. Ceci signifie que dans les 3 structures de boucles décrites ci-dessus, le code à exécuter dans la boucle peut lui-même comporter une boucle. Ce processus d'imbrication peut se répéter autant que nécessaire.

Nous mettrons en œuvre les boucles imbriquées par la suite, par exemple dans la manipulation de tableaux, et lors de procédures de tris.

3.) Instructions de contrôle des boucles

L'instruction break

L'instruction break permet de sortir d'une structure de boucle avant que la condition du test soit respectée. Quand la boucle rencontre une instruction break, elle se termine immédiatement en ignorant le code restant. En voici un exemple :

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
int nx = 0;
while (nx < 10)
{
    System.out.println("sucre");
    nx++;
    if (nx == 5)
        break;
    else
        //faire quelque chose d'autre
}
```

Dans cet exemple, la boucle s'arrête quand nx est égal à 5.

L'instruction continue

L'instruction `continue` permet d'ignorer le reste de la boucle et de reprendre l'exécution à l'itération suivante de la boucle.

```
int nx = 0;
for (nx = 0 ; nx < 10 ; nx++)
{
    if(nx == 5)
        continue;
    //revient au début de la boucle avec nx=6
    System.out.println("lait");
}
```

Cet exemple n'imprime pas "lait" si x a la valeur 5.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

VI.] LES TABLEAUX

1.) Déclaration d'un tableau

Un tableau est une structure de données qui peut contenir plusieurs éléments du même type. Un tableau d'éléments se déclare comme suit :

```
<type élément> [ ] <nom du tableau> ;
```

Ou comme suit

```
<type élément> <nom du tableau> [ ] ;
```

Les éléments d'un tableau peuvent être de n'importe quel type : type de base, type composite ou classe définie par l'utilisateur. Examinons quelques exemples de déclarations de tableaux :

```
int idEtudiant[];  
char[] cNiveaux;  
float dCoordonnées[][];
```

Ces déclarations amènent deux remarques :

- La taille du tableau n'est pas précisée dans la déclaration.
- Les crochets peuvent suivre l'identificateur, comme dans le premier exemple, ou suivre le type de données, comme dans le deuxième exemple.

2.) Création et initialisation des tableaux

Les déclarations de tableaux ci-dessus n'affectent pas de mémoire aux tableaux (elles déclarent simplement des identificateurs qui permettront éventuellement de stocker des tableaux réels). C'est la raison pour laquelle les tailles des tableaux ne sont pas précisées.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Pour affecter réellement de la mémoire aux variables tableaux, il faut utiliser l'opérateur **new** de la façon suivante :

```
<nom du tableau> = new <type élément> [<taille réservée>];
```

Par exemple :

```
idEtudiant = new int[20];  
cNiveaux = new char[5];  
dCoordonnées = new float[10][5];
```

Il est possible de combiner déclaration et réservation d'espace en une seule instruction :

```
<type élément> <nom du tableau> = new <type élément> [<taille réservée>];
```

Comme par exemple :

```
int idEtudiant[] = new int[20];  
char[] cNiveaux = new char[20];  
float[][] dCoordonnées = new float[10][5];
```

La première instruction crée un tableau de 20 éléments de type int, la deuxième crée un tableau de 20 éléments de type char et la troisième crée un tableau à deux dimensions 10 sur 5 de type float (10 lignes, 5 colonnes).

Quand un tableau est créé par new(), aucun élément n'est initialisé.

Pour initialiser un tableau, les valeurs des éléments du tableau peuvent être énumérées à l'intérieur d'un ensemble entre accolades. Pour les tableaux à plusieurs dimensions, il faut utiliser des accolades imbriquées. Les instructions suivantes illustrent cette initialisation :

```
char[] cNiveaux = {'A', 'B', 'C', 'D', 'F'};  
float[][] dCoordonnées = {{0.0, 0.1}, {0.2, 0.3}};
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

La première instruction crée un tableau de type `char` appelé `cNiveaux`. Elle initialise les éléments du tableau avec des valeurs comprises entre 'A' et 'F'. Nous n'avons pas eu besoin d'utiliser l'opérateur `new()` pour créer ce tableau ; en initialisant le tableau, la mémoire nécessaire est automatiquement affectée au tableau pour y mettre les valeurs d'initialisation. Par conséquent, la première instruction crée un tableau de type `char` de 5 éléments.

La deuxième instruction crée un tableau de type `float` à deux dimensions appelé `dCoordonnées`, avec une taille de 2 sur 2. La première ligne du tableau est initialisée avec les valeurs 0.0 et 0.1 et la deuxième ligne avec les valeurs 0.2 et 0.3. De par sa conception, `dCoordonnées` est un tableau contenant deux éléments tableau.

3.) Accès aux éléments des tableaux

Pour accéder aux éléments d'un tableau, il faut **indexer** la variable tableau. Cette indexation implique que le nom de la variable tableau soit suivi du numéro de l'élément (index) entre crochets ([et]). L'**index** des tableaux **commence toujours à 0**. Dans le cas d'un tableau à plusieurs dimensions, il faut utiliser un index par dimension.

Voici quelques exemples :

```
premierElément = cNiveaux[0];           //premierElément = 'A'
cinquièmeElément = cNiveaux[4];         //cinquièmeElément = 'F'
ligne2Col1 = dCoordonnées[1][0];       //ligne2Col1 = 0.2
```

La petite partie de code suivante illustre l'utilisation des tableaux. Elle crée un tableau de 5 éléments de type `int` appelé `tableauInt`, puis utilise une boucle `for` pour mettre les nombres entiers 0 à 4 dans les éléments du tableau :

```
int[] tableauInt = new int [5];
int index;
for (index = 0; index < 5; index++)
    tableauInt[index] = index;
```

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Ce code utilise la boucle `for` pour incrémenter la variable `index` de 0 à 4 et, à chaque passage, la valeur de `index` est mise dans l'élément de `tableauInt` indexé par `index`.

4.) Taille d'un tableau

La taille des tableaux est fixe. Une fois initialisée, elle ne peut plus être modifiée. Cependant, il n'est pas nécessaire qu'elle soit connue au moment de la compilation. Un tableau peut être créé de façon dynamique, pendant l'exécution du programme, comme le montrent les instructions suivantes:

```
int[] tableauInt;  
int nTaille;  
.  
.  
.  
nTaille = 7;  
tableauInt = new int[nTaille];
```

Tous les tableaux possèdent un champ `length` qui peut être interrogé pour connaître leur nombre d'éléments. L'instruction suivante affiche la taille du tableau :

```
System.out.println(tableauInt.length);
```

Le champ `length` des tableaux est souvent utilisé pour initialiser ses éléments à l'aide d'une boucle `for`.

```
for (index = 0; index < tableauInt.length; index++)  
    tableauInt[index] = 0;
```

5.) Références de tableaux

Il est possible de recopier une variable de type tableau dans une autre, comme dans les lignes suivantes

```
class RefTableau  
{  
    public static void main(String args[])  
    {  
        char[] cNiveaux = {'A', 'B', 'C', 'D', 'E'};  
        char[] cTableau = {'Z', 'Y', 'X', 'W', 'V'};  
        int i;  
  
        cTableau = cNiveaux;
```

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

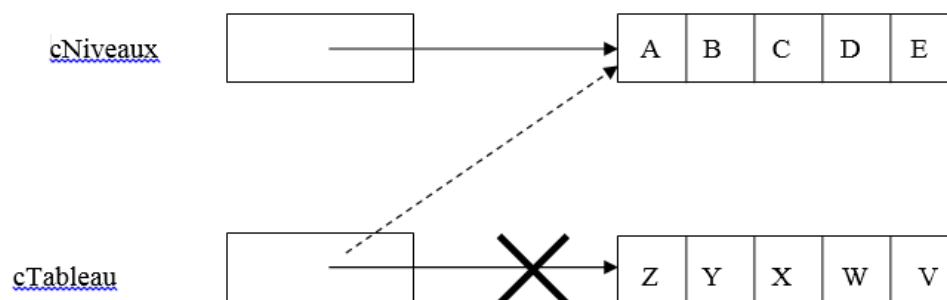
INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

```
for (i = 0; i < 5; i++)  
    System.out.print(cNiveaux[i]);  
System.out.println();  
for (i = 0; i < 5; i++)  
    System.out.print(cTableau[i]);  
System.out.println();  
cNiveaux[3]='R';  
System.out.println(cTableau[3]);  
}  
}
```

Cependant, l'instruction `cTableau = cNiveaux;` signifie que `cTableau` référence le même tableau que `cNiveaux`. Ceci est démontré dans la suite du programme : modifier l'élément de rang 3 de `cNiveaux` revient à modifier l'élément de rang 3 de `cTableau` !

L'instruction `cTableau = cNiveaux;` peut se représenter comme suit :



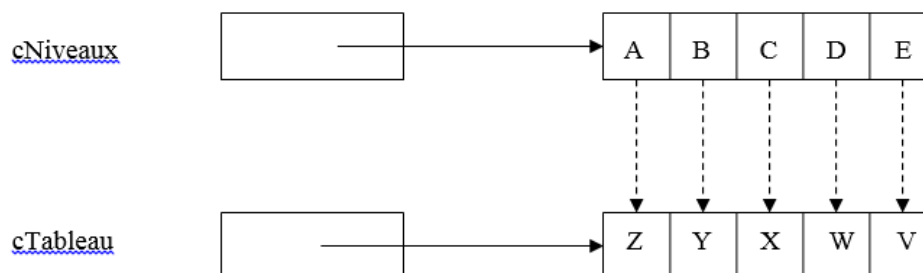
Cette instruction ne correspond pas à la copie des éléments de `cNiveaux` dans ceux de même rang de `cTableau`.

FORMATION AFPA
- CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA
PROGRAMMATION PROCEDURALE AVEC JAVA

Pour obtenir une telle recopie, qui est illustrée à la figure suivante :



Il faudrait recopier un à un, à l'aide d'un boucle, les éléments de cNiveaux dans ceux de cTableau.

6.) Les chaînes de caractères

En Java, les chaînes de caractères sont des objets, instances de la classe String. Ce concept sera abordé plus tard. Néanmoins, grâce aux tableaux, nous pouvons déjà gérer des chaînes de caractères rudimentaires. Dans ce langage, une chaîne de caractères est mémorisée dans un tableau de caractères de taille fixe ; il est nécessaire de pouvoir gérer n'importe quelle chaîne. Par conséquent

- elle doit « tenir » dans le tableau (éviter le débordement du tableau)
- comme elle occupe éventuellement une place moindre que la taille fixe du tableau, la fin de la chaîne est marquée par un caractère spécial '\0'

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA PROGRAMMATION PROCEDURALE AVEC JAVA

VII.] LES FONCTIONS

1.) Introduction

Une fonction est un ensemble d'instructions possédant un nom, susceptible d'être utilisé plusieurs fois au sein du programme même, ou à partir d'autres programmes.

En Java, les fonctions ne peuvent être définies qu'à l'intérieur d'une classe.

L'environnement de développement Java comporte de nombreuses classes déjà écrites, et réparties dans des paquetages (*packages*). Ceux-ci sont structurés de façon arborescente. Certaines de ces fonctions sont accessibles directement à partir de n'importe quel programme java : c'est le cas du package `java.lang`, qui contient « les classes de bases du langage » : la classe `System` qui sert aux entrées/sorties standards, la classe `Math`, la classe `String`, ...

Pour les autres packages ou classes, il suffit de les importer dans notre programme pour pouvoir les utiliser. Par exemple, pour importer la classe `InputStream` du package `java.io`, il suffit de l'instruction :

```
import java.io.InputStream;
```

Pour importer toutes les classes du même package, voici l'instruction

```
import java.io.*;
```

Dans un premier temps, il ne sera pas nécessaire de faire des imports.

2.) Les fonctions prédéfinies

La classe `Math`

La classe `Math` illustre le concept de fonctions prédéfinies : elle fournit des fonctions qui implémentent des fonctions mathématiques courantes. Parmi celles-ci, se trouvent : `sin`, `cos`, `exp`, `log`, `max`, `min`, `random`, `sqrt` et `tan`.

FORMATION AFPA - CONCEPTEUR DEVELOPPEUR INFORMATIQUE -

(NIVEAU II)

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Voici quelques exemples d'utilisation de ces fonctions.

```
double d1 = Math.sin(45);  
double d2 = 23.4;  
double d3 = Math.exp(d2);  
double d4 = Math.log(d3);  
double d5 = Math.max(d2, Math.pow(d1, 10));  
double d6 = Math.random();
```

- Une fonction possède un nom spécifique. L'appel (ou l'exécution) d'une fonction se fait en écrivant, dans une instruction, le nom de la fonction, suivi de parenthèses ; les paramètres éventuels de cette fonction sont indiqués entre ces parenthèses. Si ces fonctions font partie d'une bibliothèque (comme par exemple Math), le nom de la fonction est préfixé du nom de la bibliothèque.
- Une fonction peut retourner un résultat ; dans le cas des fonctions de la bibliothèque Math, toutes les fonctions retournent une valeur double, comme par exemple celle qui calcule la racine carrée d'une valeur. Pour mémoriser la valeur retournée par la fonction, il suffit d'écrire une instruction d'affectation, dans laquelle la fonction est indiquée à droite du signe = ; la variable située à gauche du signe = reçoit la valeur de retour de la fonction après que celle-ci ait été exécutée.
- Une fonction possède 0, 1, 2, ... paramètres. Ainsi,
 - la fonction Math.random() ne possède aucun paramètre : elle retourne un nombre aléatoire de type double compris entre 0.0 et 1.0
 - la fonction Math.sin() possède un seul paramètre : elle retourne le sinus de l'angle passé en argument (exprimé en radian)
 - la fonction Math.max() possède 2 paramètres : elle retourne le plus grand des 2 nombres passés en paramètres
- Lorsqu'une instruction mentionne l'appel d'une fonction, il faut impérativement écrire les parenthèses (et), même si la fonction ne possède aucun paramètre.
- Lors de l'appel d'une fonction il faut **respecter le type et l'ordre des paramètres**

En outre, la classe Math déclare les constantes PI et E qui peuvent être utilisées dans des calculs.

INITIATION AU LANGAGE JAVA

3.) Les fonctions personnalisées

Le langage Java permet au programmeur de développer ses propres fonctions. Ainsi, lorsque le problème à étudier révèle la nécessité de faire appel plusieurs fois à un même ensemble d'instructions, avec éventuellement un ou plusieurs paramètres qui modifient le résultat obtenu, la mise en œuvre de fonctions constitue une solution efficace.

Dans ce chapitre, seules seront abordées les fonctions *static* (méthodes de classe : voir explication plus tard) ; ces fonctions seront impérativement déclarées *public static*.

Pour pouvoir mettre en œuvre les fonctions, il faut les implémenter (définir) et les appeler.

Implémentation d'une fonction

Implémenter une fonction correspond à définir les instructions qui seront exécutées lorsque la fonction sera appelée. Plus précisément, il faut

- Définir les instructions qui composent la fonction
- Attribuer un nom à cette fonction
- Déterminer les paramètres d'entrée nécessaires à l'exécution de la fonction (appelés **paramètres formels** : il s'agit de préciser leur type et leur nom)
- Déterminer la valeur de sortie de la fonction (résultat retourné par la fonction : il suffit de préciser son type). Si la fonction ne retourne rien, elle est de type `void`.

Les syntaxes d'implémentation d'une fonction sont les suivantes :

- Si la fonction retourne un résultat :

```
public static <type retour> nomFonction (<type paramètre> <paramètre>, ...)  
{  
    ...  
    // bloc d'instructions de la fonction  
    <type retour> result;  
    ...  
    result = ... ;  
    return result;  
}
```

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Pour préciser les paramètres il suffit de faire suivre le nom de la fonction par la liste, entre parenthèses, et séparée par des virgules, de couples <type de paramètre> < paramètre formel>

Si la fonction ne retourne aucun résultat :

```
public static void nomFonction (<type paramètre> <paramètre>, ...)  
{  
    ...  
    // bloc d'instructions de la fonction  
}
```

Dans la syntaxe ci-dessus, les paramètres formels déclarés entre parenthèses sont des variables **locales** à la fonction. Dès que la dernière instruction de la fonction est terminée, les variables locales à la fonction sont hors de portée (elles n'existent plus).

L'instruction return

Dans le cas où la fonction retourne une valeur, l'instruction *return* met cette valeur à la disposition de la fonction appelante ; celle-ci peut exploiter cette valeur, par exemple par une affectation à une variable de la fonction appelante.

Appel d'une fonction

Appeler une fonction, c'est

- utiliser le nom de la fonction dans une instruction. Si la fonction appelée est membre de la classe dans laquelle elle est utilisée, il suffit de mentionner son nom ; si ce n'est pas le cas, il faut préfixer le nom de la fonction par celui de la classe dont elle est membre.
- éventuellement préciser des valeurs pour les paramètres d'entrée (appelées **paramètres réels** ou **paramètres effectifs**)
- éventuellement exploiter le résultat fourni par cette fonction (par exemple dans une affectation) dans le programme appelant

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Exemple1

Ecrivons une fonction qui calcule et retourne le carré d'un nombre de type double.

Implémentation :

Instructions : multiplier la valeur du paramètre par la valeur du paramètre

Nom : calculeCarre()

Paramètre d'entrée : dd type double

Paramètre de sortie : type double

Appel :

```
double dResult;  
dResult = calculeCarre(3);           // appel correct  
System.out.println(calculeCarre(3)); // appel correct  
calculeCarre(3);                     // appel incorrect
```

Ceci nous donne :

```
public class Exemple1  
{  
    public static void main (String[] args)  
    {  
        double dPar = 12.0;  
        double dCarre;  
        dCarre = calculeCarre(dPar);  
        System.out.println(dCarre);  
    }  
  
    public static double calculeCarre(double dd)  
    {  
        double result;  
        result = dd*dd;  
        return result;  
    }  
}
```

Lors de l'appel de fonction `calculeCarre(dPar)`, le contrôle est transféré à la fonction `calculeCarre()`; dans celle-ci, la variable `dd` prend la valeur de `dPar`, et les instructions de la fonction `calculeCarre()` sont exécutées, jusqu'au `return result`, qui renvoie à la fonction appelante la valeur du résultat.

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Exemple2

Ecrivons une fonction qui calcule et affiche le carré d'un nombre de type double.

Implémentation :

Instructions : multiplier la valeur du paramètre par la valeur du paramètre

Nom : calculeCarre()

Paramètre d'entrée : dd type double

Paramètre de sortie : aucun

Appel :

```
double dResult;  
dResult = calculeCarre(3);           // appel incorrect  
System.out.println(calculeCarre(3)); // appel incorrect  
calculeCarre(3);                     // appel correct
```

Ceci nous donne :

```
public class Exemple2  
{  
    public static void main (String[] args)  
    {  
        double dPar = 12.0;  
        double dCarre;  
        calculeCarre(dPar);  
    }  
  
    public static void calculeCarre(double dd)  
    {  
        double result;  
        result = dd*dd;  
        System.out.println(result);  
    }  
}
```

Lors de l'appel de fonction `calculeCarre(dPar)`, le contrôle est transféré à la fonction `calculeCarre()` ; dans celle-ci, la variable `dd` prend la valeur de `dPar`, et les instructions de la fonction `calculeCarre()` sont exécutées ; cette fonction ne possède pas d'instruction `return`, puisqu'elle est `void`. Après affichage du résultat, le contrôle est rendu au programme appelant.

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

4.) Passage de paramètres d'entrée par valeur

La première façon de procéder consiste à passer à la fonction la valeur du paramètre dont elle a besoin. Considérons l'exemple d'une fonction qui prend pour paramètre une valeur entière et affiche son carré. Nous pourrions écrire ce programme de la façon suivante :

```
public class Fonc1
{
    public static void main (String[] args)
    {
        int nombre = 12;
        System.out.println(nombre);
        calculeCarre(nombre);
        System.out.println(nombre);
    }

    public static void calculeCarre(int n)
    {
        n = n * n;
        System.out.println(n);
    }
}
```

Ce programme affiche :

```
12
144
12
```

Lors de l'appel de la fonction `calculeCarre`, la valeur de `nombre` est passée à la fonction qui utilise cette valeur pour initialiser la variable `n`. Cette variable n'a rien à voir avec la variable `nombre`, si ce n'est qu'elle a temporairement la même valeur. À la deuxième ligne de la fonction, la valeur de `n` est modifiée. Cela ne modifie en rien la valeur de `nombre`. Lorsque la fonction retourne, nous constatons (en l'affichant) que cette variable n'a pas été modifiée. Il s'agit ici d'un paramètre passé **par valeur**, car seule la valeur de la variable est passée, et non la variable elle-même.

En résumé, pour les **types de base**, le passage de paramètres à la fonction appelée se fait par **valeur**.

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

5.) Passage de paramètres d'entrée par référence

La deuxième façon de procéder consiste à passer à la fonction l'adresse du paramètre dont elle a besoin. Considérons l'exemple d'une fonction qui prend pour paramètre un tableau de valeurs entières et affiche la moyenne des éléments du tableau. Passer tous les éléments du tableau à la fonction ne semble ni efficace ni immédiat.

Pour les variables de **type tableau**, (ou **de type objet**, voir plus tard), le passage de paramètres à la fonction appelée se fait **par référence**, c'est-à-dire par adresse : la fonction récupère, dans le paramètre formel, une référence sur le tableau passé en paramètre. Prenons un exemple

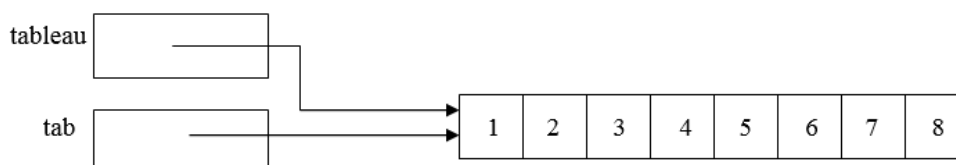
```
class Reference
{
    public static void main(String args[])
    {
        int tab[]={1,2,3,4,5,6,7,8};
        System.out.println(calculeMoyenne(tab,8));
    }

    public static double calculeMoyenne(int tableau[],
        int nb)
    {
        int i;
        double dMoyenne=0.0;
        for (i=0;i<nb;i++)
            dMoyenne+=tableau[i];
        return dMoyenne/i;
    }
}
```

Ce programme affiche

4.5

Le paramètre formel tableau, et le paramètre réel tab, constituent 2 références sur le même tableau :



INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Ce mode de passage de paramètres présente des dangers : si dans la fonction appelée, des instructions modifient le tableau passé en référence, ces modifications se répercutent immédiatement dans le programme appelant. Illustrons-le par un exemple

```
class Reference1
{
    public static void main(String args[])
    {
        int tab[]={1,2,3,4,5,6,7,8};
        int i;

        System.out.println("avant l'appel de fonction");
        for (i=0;i<8;i++)
        {
            System.out.print(tab[i]);
            System.out.print('\t');
        }
        System.out.println();
        System.out.println(calculeMoyenne(tab,8));
        System.out.println("apres l'appel de fonction");
        for (i=0;i<8;i++)
        {
            System.out.print(tab[i]);
            System.out.print('\t');
        }
        System.out.println();
    }

    public static double calculeMoyenne(int tableau[],
        int nb)
    {
        int i;
        double dMoyenne=0.0;
        for (i=0;i<nb;i++)
        {
            dMoyenne+=tableau[i];
            tableau[i]++;
        }
        return dMoyenne/i;
    }
}
```

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Ce programme fournit une moyenne correcte, mais la fonction a modifié le tableau passé en paramètre, comme le montre l'affichage

```
avant l'appel de fonction
1  2  3  4  5  6  7  8
4.5
apres l'appel de fonction
2  3  4  5  6  7  8  9
```

6.) Paramètre de retour

Pour retourner un résultat au programme appelant, une fonction dispose également de deux modes, comme dans le passage de paramètres en entrée:

- si le résultat est d'un type de base, ce résultat est retourné à l'appelant par valeur. Dans l'exemple ci-dessus, la fonction `calculeCarre()` renvoie son résultat par valeur.
- si le résultat est de type tableau ou objet, ce résultat est retourné à l'appelant par référence

7.) Portée des variables

Les règles de portée déterminent où est reconnue une variable dans un programme.

Les variables appartiennent à deux catégories de portée principales :

- Variables globales : Variables reconnues tout au long du programme, c'est-à-dire partout dans la classe. Ces variables sont déclarées en dehors de toute fonction.
- Variables locales : Variables reconnues uniquement dans le bloc de code où elles sont déclarées. Par exemple, une variable définie dans le bloc d'une fonction est locale à cette fonction (reconnue uniquement dans cette fonction)

Les règles de portée sont étroitement liées aux blocs de code. Voici la règle de portée générale : une variable déclarée dans un bloc de code n'est visible que dans ce bloc et dans les blocs qui y sont imbriqués.

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Le code suivant illustre cette règle :

```
class DemoPortee
{
    public static int nx = 0;
    public static void fonction1()
    {
        int ny;
        ny = nx;  //correct
    }
    public static void fonction2()
    {
        int nz = 1;
        nz = ny;  //incorrect !
    }
    ...
}
```

Ce code déclare une classe appelée `DemoPortee`, qui contient deux fonctions, `fonction1()` et `fonction2()`. La classe elle-même est considérée comme étant le bloc de code principal et les deux fonctions sont ses blocs imbriqués.

La variable `nx` est déclarée dans le bloc principal et elle est donc visible (reconnue par le compilateur) dans la fonction `fonction1()` et la `fonction2()`. D'autre part, les variables `ny` et `nz`, sont déclarées dans deux blocs imbriqués mais indépendants ; ainsi, essayer d'utiliser `ny` dans `fonction2()` provoque une erreur, puisque `ny` n'est visible que dans son bloc.

Un programme qui utilise des variables globales présente des risques d'erreurs car:

- les variables globales sont difficiles à suivre dans un programme complexe
- une modification apportée à une variable globale dans une partie du programme peut avoir un effet indésirable dans une autre partie du programme.

Les variables locales sont plus sûres, puisqu'elles ont une durée de vie limitée. Par exemple, une variable déclarée dans une fonction n'est accessible qu'à partir de cette fonction et ne risque pas d'être utilisée de façon erronée ailleurs dans le programme.

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

8.) Fonctions récursives

Comme la plupart des langages récents, Java permet d'écrire des fonctions récursives.

Une fonction récursive est une fonction dont une au moins des instructions est un appel à la fonction elle-même (en général avec des valeurs de paramètres différents).

Cette façon de programmer repose sur un principe de récurrence. Soit une fonction possédant un paramètre n entier. Pour écrire le code de la fonction de paramètre n , une des instructions sera un appel à la fonction avec $n-1$ comme valeur de paramètre

```
public static int maFoncRecursive(int n)
{
    ...
    // instructions
    ... maFoncRecursive(n-1);
    return ...
}
```

L'appel à la fonction avec un paramètre valant $n-1$ provoquera un appel à la fonction avec une valeur $n-2$, et ainsi de suite : les appels s'empilent ! Pour éviter un cycle d'appel sans fin, il faut que l'algorithme permette un retour pour une valeur particulière :

```
public static int maFoncRecursive(int n)
{
    if (n == <valeur particulière>)
    {
        // instructions particulières
        return ... ;
    }
    else
    {
        // instructions
        ... maFoncRecursive(n-1);
        return ... ;
    }
}
```

INITIATION AU LANGAGE JAVA

PROGRAMMATION PROCEDURALE AVEC JAVA

Exemple

Ecrivons la classe Factorielle, afin de calculer la factorielle d'un nombre, notée $n!$, avec

$$n! = n * (n-1) * (n-2) * \dots * 2 * 1$$

Ecrivons-le sous forme récurrente :

$$n! = n * (n-1)!$$

$$1! = 1 \quad (\text{valeur particulière})$$

```
class Factorielle
{
    public static void main(String args[])
    {
        long nL;
        Scanner sc = new Scanner(System.in);

        System.out.println("entrer un nombre entier");
        nL = sc.nextLong();
        System.out.println(fact(nL));
    }

    public static long fact(long n)
    {
        if (n==1)
        {
            return 1;
        }
        else
        {
            return n*fact(n-1);
        }
    }
}
```