

Chapitre I : Eléments de base du langage Java

1. Introduction

Comme tout langage de programmation, java offre des éléments pour déclarer des variables et pour les manipuler. Ce chapitre présente les éléments de déclarations pour les types primitifs et les tableaux ainsi que les instructions de base du langage. Il faut noter que de fortes ressemblances existent entre les langages C et java. Les différences seront mises en évidence au fur et à mesure de l'avancement du cours.

2. Structure d'un programme java

Un programme java est un ensemble d'*objets* créés à partir de *classes* qui les décrivent. Ces objets doivent subir des opérations via des *méthodes* définies dans ces classes. Le concept de *méthode* dans un langage orienté objet (tel que C++ ou java) correspond à la notion de fonction en programmation classique (Pascal, C,...). Une méthode est donc, une portion de programme manipulant des variables, des attributs d'objets via des instructions du langage (ou des appels à d'autres méthodes). Les concepts d'*objet*, de *classe*, de *méthode*, ... seront vus dans le chapitre 2 du cours.

Le point d'entrée de l'exécution d'un programme java est la méthode spécifique *main()* (comme en langage C), elle doit être contenue dans une classe représentant le programme principal. La syntaxe d'écriture est :

```
class programme
{ public static void main (String [ ] arg)
    { // déclarations
      // instructions
    }
}
```

class : c'est un mot réservé du langage indique que l'on commence la description d'une classe, nécessairement suivi par le nom de la classe (sur l'exemple, le nom de la classe est *programme*). Cette première ligne s'appelle l'*en-tête* de la classe.

Après l'en-tête, vient le *corps* de la classe qui est un bloc encadré entre accolades. Sur l'exemple, il y a une seule méthode (la méthode principale *main*).

public static : ces mots réservés sont appelés modificateurs et seront expliqués plus tard.

void : toute méthode peut renvoyer une valeur, le type de la valeur doit être indiqué avant le nom de la méthode (comme en C). Le type void indique que la méthode ne renvoie pas de valeur.

main: constitue le point d'entrée de l'exécution du programme. Elle admet nécessairement le même en-tête **public static void main (String [] arg)**, seul l'identificateur *arg* peut changer.

Toute méthode doit être définie à l'intérieur d'une classe, même la méthode **main**.

String: il s'agit d'une classe définie dans le package (bibliothèque de classes) java.lang servant à traiter les chaînes de caractères. La méthode main possède comme paramètres un tableau de chaînes de caractères appelé *arg*.

3. Eléments du langage

3.1 Les commentaires

Comme en langage C, les commentaires s'étalant sur une ou plusieurs lignes sont encadrés entre les symboles `/*` et `*/`. Les commentaires portant sur une seule ligne peuvent être précédés du symbole `//` (double-slash).

3.2 Affichage de données

L'affichage à l'écran en java utilise l'instruction **System.out.print** (ou **System.out.println**) définie dans le package (bibliothèque) système standard. Sa syntaxe est :

```
System.out.print ("message"); // pour afficher le texte d'un message à l'écran
System.out.print (var);    // pour afficher le contenu de la variable var
```

Pour concaténer le texte d'un message et la valeur d'une variable à l'affichage, on utilise le symbole `+`. On écrira alors :

```
System.out.print ("message"+ val);
```

Remarque: l'écriture de l'instruction peut paraître lourde mais chaque partie a une signification. En effet, *print* (ou *println*) est une méthode s'appliquant sur l'objet *out* appartenant à la classe *System* prédéfinie. Cet objet est destiné à l'affichage sur écran.

Exemple

Voici un exemple de programme qui affiche un message à l'écran

```
class Message
{ public static void main (String [ ] arg)
  { System.out.println (" ceci est un cours de Java" ); }
}
```

Autre Exemple

Voici un exemple de programme qui initialise des variables et affiche leur contenu à l'écran.

```
class prog_simple
{ public static void main (String [ ] arg)
  { int x = 20 ; float y = 1.5 ;    // déclarations et initialisations
    // affichages
    System.out.println ("la valeur de x est" + x );
    System.out.println ("la valeur de y est" + y );
  }
}
```

3.3 Déclaration de variables

La déclaration d'une variable se fait selon la syntaxe :

```
Type nom_variable;
```

Remarque : la syntaxe de déclaration est semblable à celle du langage C. Il est aussi possible d'initialiser une variable à la déclaration.

➤ Les types primitifs en java

Ces types sont semblables à la plupart des types dans les langages de programmation, il s'agit des types tels que booléen, entier, caractère, réel, ...

L'orientation objet du langage fait qu'à chaque type primitif est associée une classe appelée *classe enveloppe*, ces classes sont définies dans le paquetage *java.lang*. Une classe enveloppe fournit des méthodes prédéfinies de manipulation du type associé.

Le tableau suivant résume les types de données *primitifs* manipulés en java ainsi que les classes « enveloppes » associées.

Type primitif	Taille	Classe enveloppe
Boolean	1 bit	Boolean
Byte	1 octet	Byte
Char	2 octets	Character
Short	2 octets	Short
Int	4 octets	Integer
Long	8 octets	Long
Float	4 octets	Float
double	8 octets	Double

Remarques : Contrairement au C, le type *booléen* n'est pas un scalaire en java, il est défini sur *1bit*, prenant deux valeurs particulières *true* et *false*.

Le type caractère (char) est codé sur 2 octets selon le standard *Unicode* et permet ainsi de coder un grand nombre de caractères différents (2^{16}).

Exemple

```
class Types_Num
{
    public static void main (String [ ] arg)
    {
        byte b; int x; short s;
        /* MIN_VALUE et MAX_VALUE sont des constantes de classes définies dans les classes
           enveloppes */
        System.out.println ("byte : [" + Byte.MIN_VALUE + ", " +
                             Byte.MAX_VALUE + "] ");
        System.out.println ("short : [" + Short.MIN_VALUE + ", " +
                             Short.MAX_VALUE + "] ");
        System.out.println ("entier : [" + Integer.MIN_VALUE + ", " +
                             Integer.MAX_VALUE + "] ");
    }
}
```

L'exécution de ce programme donnera les affichages suivants:

byte : [-128, 127]

short : [-32767, 32768]

entier : [-2147483648, 2147483647]

➤ Conversions de types lors d'affectations

On peut affecter une variable de type *char* à une variable de type *int* ou *long*. Les autres affectations d'une expression de type *int* à une variable de type *char* ou d'une variable de type *char* à une variable de type *short* ou *byte* peuvent se faire moyennant un transtypage.

On pourra écrire :

```
int x; char c = 'a'; short s;
x = c; s = (short) c ; // on convertit le char en short
```

L'affectation d'un type plus petit dans un type plus grand est possible sans problème. En effet, un byte peut être affecté à un short, int, long,... Un short peut être affecté à un int, long, ... Un int peut être affecté à un long, float, double. Un float peut être affecté à un double.

On pourra écrire : int x = 5; float y = x; long z = x;

Dans le sens inverse, il est possible d'affecter un type plus grand dans un type plus petit moyennant un transtypage (imposer un casting) mais souvent, il ya perte d'information pour les nombres dépassant la taille du type receveur.

On pourra écrire : int x ; float y = 4.5 ; float z = 9.9999998E¹⁰ ;
 x = (int) y; long lg = (long) z ;
 System.out.println ("x =" + x + " et lg =" + lg);

L'affichage donnera dans ce cas : x = 4 et lg = 9999997952

3.4 Les instructions de base

➤ L'affectation

L'affectation se fait à l'aide du symbole = comme en C

Nom_variable = expression ;

Les expressions: on distingue deux types d'expressions, les expressions arithmétiques et les expressions logiques.

Les expressions arithmétiques : sont définies à l'aide d'opérandes (variables ou valeurs) et d'opérateurs arithmétiques. Ces opérateurs sont identiques à ceux du langage C (+ pour la somme, - pour la différence, % pour le modulo, * pour la multiplication et / pour la division) avec les mêmes règles de priorité qu'en C.

On manipule aussi les opérateurs de pré et post-incrémentation, de pré et post-décrémentation (++i, i++, --i, i--). Ainsi que les opérateurs +=, -=, *= et /=

Les expressions logiques : ne peuvent prendre que les valeurs *true* ou *false* et sont définies à l'aide d'opérandes (variables ou valeurs) et d'opérateurs logiques (<, >, >=, <=, ! (not), != (différent), == (égal), && (et), || (ou)) identiques à ceux définis dans le langage C avec les mêmes règles de priorité.

➤ Instructions conditionnelles

Instruction if

```
if (<cond>) instr_1 ; else instr_2;  
                  ou  
if (<cond>) {instr_1 ; ...;instr_n}; else {instr_1; ...instr_m};
```

On peut avoir des conditions imbriquées

```
if <cond1> instr_1 ;  
  else if <cond2> instr_2;  
      else instr_3;
```

Instruction switch

```
switch (i)  
{ case 1 : instructions ;
```

```

        break ;

    case 2 : instructions;
        break;
    default: ...
}
i: variable de contrôle (sélecteur du switch)
break : fait sortir du switch

```

➤ Instructions itératives

Instruction while

```

while (cond) instr ;
ou
while (cond) {instr_1 ;... ; instr_n ;}

```

Instruction do... while

```

do instr ; while (cond) ;
ou
do {instr_1; ...; instr_n;} while (cond);

```

Instruction for

```

for (expr1 ; expr2 ; expr3) instr ;
ou
for (expr1 ; expr2 ; expr3) {instr_1 ;...instr_n ;}

```

expr1 : valeurs de départ (plusieurs initialisations possibles, séparés par des virgules)
expr2 : condition pour effectuer les instructions de la boucle.
expr3 : instruction à effectuer à chaque fin de boucle (plusieurs instructions possibles séparées par des virgules).

➤ Les instructions break et continue

L'instruction **break** permet de sortir directement d'une boucle (for, while ou do...while).
 L'instruction **continue** permet d'interrompre une itération et passer directement à l'itération suivante.

4. Saisie de données

La saisie de données en java peut se faire en utilisant la classe **Scanner** définie dans le package java.util, selon la syntaxe :

```
Scanner e = new Scanner (in); /* creation d'un objet e de la classe Scanner
```

Pour lire un entier x au clavier, on écrira :

```
int x = e.nextInt();
```

Pour lire un réel y, on écrira :

```
float y = e.nextFloat();
```

Pour lire une chaîne de caractères s, on écrira :

```
String s = e.nextLine() ; ou String s = e.next() ;
```

5. Les tableaux

Un tableau est une suite de composantes (éléments) destinées à contenir des données de même type. En java, les tableaux ne sont pas vraiment des objets et sont encore moins des types primitifs (ils sont situés entre les deux). Néanmoins, un tableau se rapproche plus d'un objet par le fait que:

- Il est manipulé par référence, la variable qui le désigne est une adresse.
- Nécessite en général une opération de création `new` pour le définir (lui allouer de la mémoire).
- Une variable de type « référence d'un Object » peut être utilisée pour référencer un tableau.

Comme en C, les éléments du tableau sont indicés à partir de 0 ; ils sont initialisés par défaut: un booléen est initialisé à `false`, un `char` est initialisé avec le caractère de valeur nulle, un entier ou réel est initialisé à 0, un type par référence est initialisé à la valeur **null (en minuscule)**.

5.1 Déclaration, création et manipulation d'un tableau

Pour définir un tableau, on peut utiliser la notation suivante :

type des éléments [] nom du tableau; ou encore
type des éléments nom du tableau [];

Exemple : `int[] tab;` ou bien `int tab[];`

Cette écriture déclare la variable **tab**, comme référence (adresse). Aucune allocation mémoire n'est faite pour les éléments du tableau. Pour allouer de l'espace au tableau, on utilise l'opérateur de création **new**, ainsi on devra écrire :

Nom du tableau = new type des éléments [nombre d'éléments];

Exemple

```
tab = new int [10] ;
```

Cette écriture crée (réservation d'espace) un tableau de 10 éléments entiers.

Voici un petit programme qui manipule un tableau d'entiers.

```
class Prog_tableau
{ public static void main (String [ ] arg )
  { int [ ] tableau;
    tableau = new int [2]; // crée un tableau de deux éléments entiers
    tableau [0] = 15;
    System.out.print ("tableau[0] = " + tableau[0] + " tableau [1] = " +
      tableau[1]); }
}
```

➤ Autre façon de déclarer un tableau

On pourrait déclarer un tableau et l'initialiser directement sans utiliser l'opérateur de création `new`. Voici un exemple qui illustre ceci en créant un tableau B de type booléen:

```
class Creer_tableau
{ public static void main (String [ ] arg )
  { boolean [ ] B = {true, false, true} ;
    // crée et initialise un tableau de trois éléments booléens
  }
```

```

        System.out.print ( " Le deuxième élément est" + B[1] ); }
    }

```

Pour connaître le nombre d'éléments d'un tableau, on peut utiliser l'attribut **length** (c'est le seul attribut défini pour un tableau), en utilisant la syntaxe :

Nom du tableau.length

Par exemple, si on reprend le tableau de booléens précédent : B. length sera égal à 3. L'indice du dernier élément de B est égal à B.length -1 (c'est-à-dire 2).

5.2 Tableau à deux dimensions (matrices)

Pour déclarer un tableau à deux dimensions, on peut procéder de deux manières :

- **La première manière** : Allouer la matrice en une seule fois en précisant les deux dimensions. Voici un exemple d'illustration :

```

class Prog_matrice
{
    public static void main (String [ ] arg )
    {
        int [ ] [ ] mat; int i, j;
        mat = new int [2] [3]; // crée un tableau de six éléments entiers
        for (i = 0 ; i < mat.length ; i++)
            for (j = 0; j < mat[i].length; j++)
                mat[i][j] = i+j;
        System.out.print ("mat[0][1] = " + mat[0][1]); }
    }

```

- **La deuxième manière** : Allouer d'abord un vecteur de références vers des tableaux à une dimension ; ainsi on aura défini un tableau pour référencer les lignes de la matrice. Puis on alloue (crée) une par une les lignes de la matrice. Les lignes ne doivent pas avoir nécessairement la même dimension. Le même exemple pourrait se réécrire comme suit :

```

class Prog_matrice
{
    public static void main (String [ ] arg )
    {
        int [ ] [ ] mat; int i, j;
        mat = new int [2] [ ] ; // crée deux éléments qui sont des références
        for (i = 0 ; i < mat.length ; i++)
            {
                mat[i] = new int [3] ; // on crée les lignes une par une
                for (j = 0; j < mat[i].length; j++)
                    mat[i][j] = i+j; }
        System.out.print ("mat[0][1] = " + mat[0][1]); }
    }

```

Si l'on veut créer une matrice avec des lignes de tailles différentes, on doit utiliser cette deuxième technique. Ainsi pour avoir la matrice avec 3 colonnes sur la première ligne et 4 colonnes sur la 2^{ème} ligne, on écrira :

```

class Prog_matrice
{
    public static void main (String [ ] arg )
    {
        int [ ] [ ] mat; int i, j;
        mat = new int [2] [ ] ; // crée deux éléments qui sont des références
        for (i = 0 ; i < mat.length ; i++)

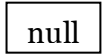
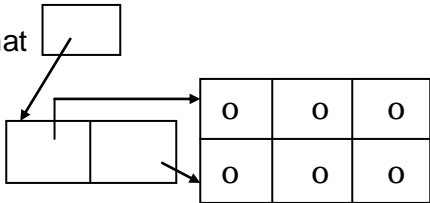
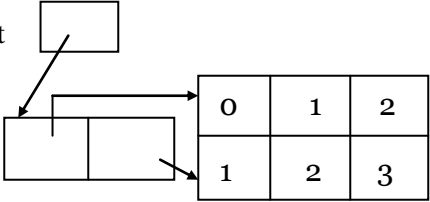
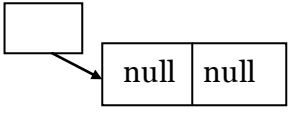
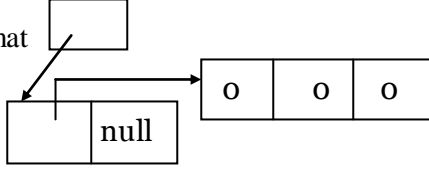
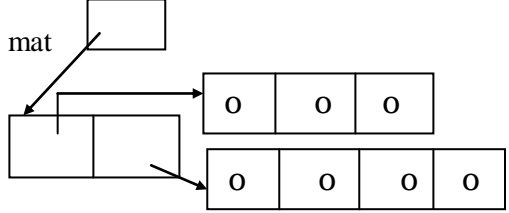
```

```

    { mat[i] = new int [3+i];
      // on crée les lignes une par une avec des tailles différentes
      for (j = 0; j < mat[i].length; j++)
        mat[i][j] = i+j; }
    System.out.print ("mat[0][1] = " + mat[0][1]); }
}

```

6. Quelques illustrations de déroulement

Déclaration/ Instruction	Etat en mémoire
<code>int [] [] mat;</code>	mat 
<code>mat = new int [2] [3];</code>	mat 
<code>for (i = 0 ; i < mat.length ; i++)</code> <code> for (j = 0; j < mat[i].length; j++)</code> <code> mat[i][j] = i+j;</code>	mat 
<code>mat = new int [2] [];</code>	mat 
<code>mat [0] = new int [3];</code>	mat 
<code>mat [1] = new int [4];</code>	mat 

Résumé

On a donc, trois sortes d'entités manipulées en java :

- Les types primitifs (entier, caractère, réel, double, booléen, entier court, octet)
- Les tableaux manipulés par référence
- Les objets (à voir au chapitre 2) manipulés par référence

Voici un tableau qui montre les similitudes et les différences entre les langages C et java

Eléments du langage	Langage C	Langage java
Structure d'un programme	Ensemble de fonctions	Ensembles de classes et d'objets
Point d'entrée de l'exécution	La fonction main	La méthode main
Type booléen	N'existe pas	Boolean
Type pointeur	Existe (* type)	N'existe pas (référence d'objet)
Chaînes de caractères	Tableau de caractères	String (objet)
Tableaux	Considérés comme des variables	Considérés comme des objets
Structures	Le type struct	La notion de classe
Opérateurs arithmétiques	+ - * / % ++, --, ...	identiques
Opérateurs logiques	&&, , !, ==, >, <, ...	Identiques
Instruction d'affichage	La fonction printf	La méthode System.out. print
Instruction de saisie	La fonction scanf	Les méthodes nextInt, nextFloat, ... de la classe Scanner . System.in. read pour les caractères
Instructions Conditionnelles	if, switch	identiques
Instructions itératives	for, while, do...while	identiques
Sorties de boucles	break, continue	identiques
Réutilisation	bibliothèque de fonctions	bibliothèque de classes