Chapitre 3

Les affectations

1 Littéraux pour les types primitifs

Les exemples suivants sont des littéraux de type primitif :
'b' // littéral char
42 // littéral int
false // littéral boolean
double k=2546789.343 // littéral double
double s= 25.36D // D ou d optionnel non obligatoire
float g= -63.25f // suffixe f ou F obligatoire après le nombre
long f= 256l; // suffixe l ou L optionnel après le nombre

1.1 Les Littéraux char

```
char a = 'a';
char b = '@';
char letterN = '\u004E'; // The letter 'N' Unicode
char b = 982;          // int literal
char c = (char)70000;  // The cast is required; 70000 is out of char range
char d = (char) -98;
char c = '\'';    //le charactère quote
 d = '\n';    //nouvelle ligne
```

1.2 Littéraux entiers

Un littéral « entier » est de type **int.** Si il est suffixé par « L » ou « l » alors elle est de type **long.**
Exemple:
108 // 108 littéral de type int,
108L // 108 littéral de type long

- Représentation en octal (8bits) et hexadécimale (16 bits).
  014 // 14 en octal (base 8) = 12 en décimal
  0xA7 // A7 en hexadécimal (base 16) = 167 en décimal
- Types des résultats des calculs avec des nombres entiers
  - tout calcul entre entiers donne un résultat de type **int**
  - si au moins un des opérandes est de type **long**, alors le résultat est de type **long**

1.3 Littéraux réels

Un littéral réel est par défaut de type double. Pour spécifier un float, il faut la suffixée par « F » ou « f ».

2 L'affectation des primitifs

Un entier littéral (exemple 7) est toujours implicitement un int.
byte b = 27; // implicit cast of 27 to a byte
est identique à:
byte b = (byte) 27; // cast explicite du littéral int au byte
Exemple:

```
byte b = 3;
byte c = 8;
byte d = b + c; // ne compile pas
byte d = (byte) (c + b);
```

2.1 Transtypage (cast) de primitifs

Exemple:
int i=13; // i codé sur 32 bit; 13 littéral int codé sur 32 bits
byte b=i; // Erreur: pas de conversion implicite int→ byte

- Pour que ça compile, il faut faire un cast (transtypage) c'est-à-dire on doit forcer le programme à changer le type de int en byte.

(*type-forcé***)** *expression*

byte b=(byte)i; // de 32 bits vers 8 bit. b vaut 13 codé sur 8 bits

- Un **cast implicite** aura lieu après affectation d'une petite chose (disons, un byte) dans un conteneur plus grand (comme un int).
  (int→ long, short → int, byte → short) et vers les types flottants.
- La conversion d'une valeur-large-dans-un-petit-conteneur est référée comme un rétrécissement et nécessite un **cast explicite**.
- Un *cast entre types primitifs peut entrainer* une perte de données sans aucun avertissement ni message d'erreur.

Exemples :

1) int i=13;

byte b=(byte)i;
b vaut 13 : conversion de 32 bits vers 8 bit. i est un entier « petit » => conversion sans perte de données

2) i=256

												1	0	0	0	0	0	0	0	0

b vaut 0 car conversion de 32 bits vers 8 bits (On considère les 8 bits de poids le plus faible).

2) int i = 130;

b = (byte)i; // b = -126 ! Pourqoui ?

												1	0	0	0	0	0	1	0

1 représente le bit de signe => Nombre négatif
Pour trouver le décimal d'un nombre négatif:
- Remplacer les bits 1 par 0 et inversement => on obtient: 01111101
- Ajouter 1 => 01111110

- Une affectation entre types *primitifs* peut utiliser un *cast* implicite si elle ne provoque aucune perte
- L'affectation peut utiliser un cast explicite (voir exemples ci-dessus)
- Les casts de types « flottants » vers les types entiers tronquent les nombres

Exemple: 1) float x=1.99;
int i = (int)x; // i = 1, et pas 2. Une troncature est faite et non un arrondi.
2) **int** x = 10, y = 3;
double z;
z=x/y; // donne z=3.0; car x/y donne 3
si on veut que z=3.3333 et pas 3.0
z = (double)x / y; // cast de x suffit

3 Notion de portée et durée de vie des objets

- Ne pas accéder à une variable d'instance dans un contexte static
- Un bloc est un ensemble d'instructions délimité par les accolades { et }
- Les blocs peuvent être emboîtés les uns dans les autres
- Utiliser une variable de bloc après la fin du bloc provoque une erreur de compilation.

Exemple:

```
int a=1, b=2;
{ //début de bloc
int x=a;
x = 2*a+b;
} //fin de bloc
```

```
x = 3; //ERREUR: x n'est pas connu ici
```

- Utiliser une variable de bloc après la fin du bloc provoque une erreur de compilation.

Exemple:

```
int a=1, b=2;
{ //début de bloc
int x=a;
x = 2*a+b;
} //fin de bloc
x = 3;
//ERREUR:
x n'est pas connu ici
```

3.1 Portée des objets

- Les variables d'instance et les objets vivent dans un espace mémoire spécial.
- Les variables locales vivent dans le stack (pile)
- La durée de vie d'une variable primitive locale à une méthode est égale à la durée de vie de la méthode
- La durée de vie d'un objet local à la méthode n'est pas égale à la durée de vie de la méthode

Exemple:

Considérons une classe nommée ClasseA et soit le bloc suivant :

```
{
ClasseA objA=new ClasseA();
} // fin de portée
```

- La référence **objA** disparaît à la fin de la portée, par contre l'objet qui a été référencé par **objA** existe toujours, **mais il reste inaccessible**.

4 Gestion de la mémoire (Garbage Collector)

- Lorsqu'on perd le contrôle sur un objet. Il persiste et il occupe de la mémoire.
- Il n'existe aucun opérateur explicite pour détruire l'objet dont on n'a pas besoin,
- Mais il existe un mécanisme de gestion automatique de la mémoire connu sous le nom de ramasse miettes (en anglais Garbage Collector).
- Un même objet peut être référencé par plusieurs variables
- Lorsqu'il n'existe plus aucune référence sur un objet, il devient candidat au ramasse miette.
- Les variables cessent de référencer un objet
  - Quand on leur affecte une autre valeur, ou null
  - Quand on quitte le bloc où elles ont été définies

Le ramasse-miettes (*garbage collector) est une* tâche qui

- Travaille en arrière-plan
- Libère la place occupée par les instances non référencées
- Compacte la mémoire occupée

IL intervient

- Quand le système a besoin de mémoire
- ou, de temps en temps, avec une priorité faible

5 Les Références des variables d'Instance

Exemple:

```
public class Livre {
  private String titre;   // référence d'instance
 public String getTitre() {
   return titre;  }
 public void afficher(){System.out.println(titre);}
  public static void main(String [] args) {
    Livre b = new Livre() ;
    String s =b.getTitre();
    System.out.println(s.length()); // erreur à l'exécution
 }
```

=> titre= null. null n'est pas similaire à une String vide (""). Une valeur null signifie que la variable de référence ne référence aucun objet.

5.1 Affectation d'une variable de référence à une autre

Exemple:
Soit une classe A:

```
class A{
int x;
}
```

Quel sera le résultat du code suivant?

```
A obj = new A();
A obj1= obj;
obj.x= 5;
System.out.println(obj.x+ " " +obj1.x);
```

Résultat : ……..

L'affectation d'une variable de référence à une autre signifie que l'objet est référencé par deux variables. La modification des attributs de l'objet pour l'une de référence entraîne sa modification pour l'autre. A l'exception des variables référençant les objets String. En Java, les objets String sont immutables; ne peuvent pas changer leur valeur.

Exemple: 1) Quel est le résultat du code suivant?

```
String x = "Java";
String y = x;
System.out.println("y string = " + y);
x = x + " Bean";
System.out.println("x string " + x + "y string = " + y);
```

Résultat : ……..

=> y n'a pas changé sa valeur malgré que la valeur de x est changée et contient: Java Bean

Que se passe-t-il lorsqu'une référence de variable String est utilisée pour modifier un String?

- une nouvelle chaîne est créée (ou une chaîne pareille existe dans le piscine (pool)), laissant la chaîne originale sans la toucher
- La référence utilisée pour modifier la chaîne est affectée au nouvel objet String

2) Que fait le code suivant ?

```
String s = "Fred";
 String t = s;
t.toUpperCase();
```

=> Créer un nouveau string "FRED" et l'abandonner. s et t réfèrent à l'ancien objet string
=> L'object String "Fred" est abandonné et non référencé

6 Variables tableau (Array)

Les éléments d'un tableau auront toujours la valeur par défaut, peu importe ou est déclaré le tableau.
Un tableau doit être déclaré et construit avant d'être utilisé.

Exemple: int[] A= new int[10];

6.1 Construire un tableau à une dimension

Exemple:

```
int [] testScores;          // Déclare un tableau d'int
testScores = new int[4]; // construire un tableau et l'affecter à testScores
Thread[] threads = new Thread[5];
```

Combien d'objets sont créés avec l'instruction précédente ?

=> …….

6.1.1 Indices du tableau

Les indices d'un tableau ***tab*** commencent à ***0*** et se terminent à (***tab.length – 1).***

6.1.2 Accès aux éléments du tableau

Soit tab un tableau.

- tab[i], avec 0≤ i ≤ (tab.length – 1), permet d'accéder à l'élément d'indice i du tableau tab.
- Java vérifie automatiquement l'indice lors de l'accès. Il lève une exception si tentative d'accès hors bornes.

Exemple:

```
tab = new int[8];
int e = tab[8]; // tentative d'accès hors bornes
/* L'exécution produit le message ArrayIndexOutOfBoundsException */
```

6.1.3 Accès à la taille du tableau

La taille est accessible par le "champ" length:

Exemple:

```
tab = new int[8];
tab[0]=5;
tab[1]=17;
tab[2]=-12;
int x = tab.length; //…….
```

6.1.4 Autres méthodes de création et d'initialisation d'un tableau

Il est possible de lier la déclaration, la création et l'initialisation explicite d'un tableau. La longueur du tableau ainsi crée est alors calculée automatiquement d'après le nombre de valeurs données

• Création et initialisation explicite à la déclaration

```
  int tab[] = {5, 2*7, 8}; // la taille du tableau est fixée à 3
  Etudiant [] etudiantsLFIM = { new Etudiant("Mohammed", "Ali"),
 new Etudiant("Fatima", "Zahra")
 } // la taille du tableau est fixée à 2
```

6.1.5 Affectation de Tableaux

• Java permet de manipuler globalement les tableaux par affectation de leurs références.
• l'affectation ne modifie que la référence.

Exemple: Soient tab1 et tab2 deux tableaux

tab1=tab2;

=> La référence de tab2 est affectée à tab1. Maintenant tab1 et tab2 désignent le même objet tableau qui était initialement référencé par tab2.

```
tab1[2]=3;       // => tab2[2] =3.
tab2[4]=6;      // => tab1[4] = 6;
```

6.2 Construire un tableau Multidimensionnel

En Java, les tableaux à plusieurs dimensions sont en fait des tableaux de tableaux.

Exemple: int[][] matrice=**new** int[5][6];

On peut également remplir le tableau à la déclaration et laisser le compilateur déterminer les dimensions des tableaux, en imbriquant les accolades :

Exemple:

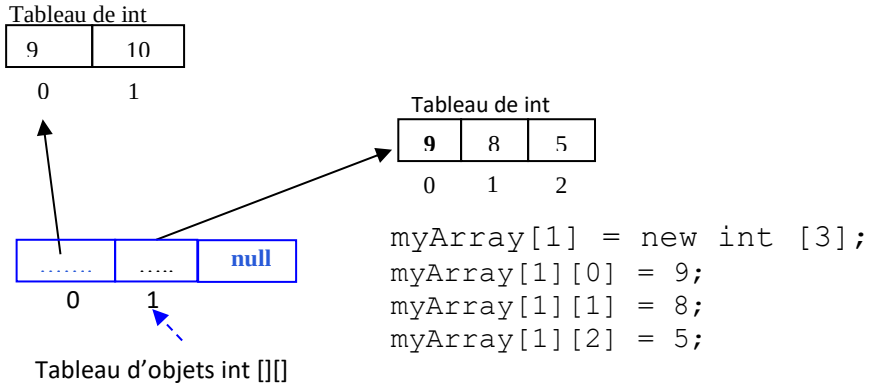
```
int[][] matrice = {
 { 0, 1, 4, 3 } , // tableau [0] de int
{ 5, 7, 9, 11, 13, 15, 17 } // tableau [1] de int
};
```

Pour déterminer la longueur des tableaux, l'attribut length est utilisé:

```
matrice.length // 2
matrice[0].length // 4
matrice[1].length // 7
```

Exemple:

```
int[] [] myArray = new int [3] [];
myArray[0] = new int [2];
myArray[0][0] = 9;
myArray[0][1] = 10;
```




```
myArray[1] = new int [3];
myArray[1][0] = 9;
myArray[1][1] = 8;
myArray[1][2] = 5;
```

6.3 Copie de tableau

La copie d'un tableau implique la copie de ses éléments dans un autre tableau. Dans le cas d'un tableau d'objets, seules les références à ces objets sont copiées, aucun nouvel objet n'est créé.

Choisir l'une des trois méthodes:

- Création d'un nouveau tableau puis copie des éléments à l'aide d'une boucle.
- Appliquer la méthode clone() de la classe Object. La valeur retournée par cette méthode est de type Object, il faut la convertir dans le type concerné.

Exemple :

```
int[] nombres = { 2, 3, 5, 7, 11 };
int[] copie = (int[]) nombres.clone();
nombres[1]=4;
// nombres contient 2 4 5 7 11
// tandis que copie contient toujours 2 3 5 7 11
```

- Copie par appel de System.arraycopy() de la classe System

System.arraycopy(src, srcPos, dest, destPos, nb);

Copie un nombre nb d'éléments du tableau src (source) à partir de l'indice srcPos et les affecte dans le tableau dest (destination) à partir de l'indice destPos.

src : tableau source

srcPos : indice de src du 1er élément copié

dest : tableau destination

destPos: indice de dst où sera copié le 1er élément

nb : nombre d'éléments copiés

Exemple:

```
import java.lang.*
System.arraycopy(tab1, 6, tab2, 2, 50);
```

Copie de 50 éléments de tab1 à partir de l'indice 6 et les affecte dans tab2 à partir de l'indice 2

6.4 Comparer deux tableaux

Comparer le contenu de deux tableaux:

1. Soit comparer un à un les éléments des deux tableaux. ;
2. Soit utiliser la méthode equals(), qui est static, de la classe Arrays. Dans ce cas, il faut importer la classe Arrays comme suit :

```
import java.util.Arrays;
```

Exemple :

```
int[] tab1,tab2;
```

```
tab1=new int[10];
tab2=new int[10];
// initialisation de tab1 et tab2
boolean b=Arrays.equals(tab1,tab2) ;
```

Remarques

➔ Les éléments d'un tableau peuvent être d'un type objet

Exemple : Considérons l'exemple de **la classe Etudiant**. Soit setCNE(String) une méthode de la classe Etudiant. Soit l'attribut : public string codeNatEtudiant

```
Etudiant [] e = new Etudiant[10];
// Chaque élément du tableau contient une référence vers un objet de type Etudiant
e[0].setCNE("11225467");
// Erreur: e[0] contient la référence vers un objet de type Etudiant.
// Il faut créer l'objet pour pouvoir l'utiliser.
e[0] = new Etudiant(); // Création de l'objet e[0]
e[0].setCNE("11225467");
```

➔ Les tableaux en argument d'une méthode: Le passage des tableaux comme paramètres des méthodes se fait par référence (comme les objets) et non par copie (comme les types primitifs). La méthode agit directement sur le tableau et non sur sa copie