Chapitre 3: Tableaux, méthodes et chaînes de caractères

Par Aïcha El Golli

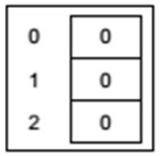
aicha.elgolli@essai.ucar.tn

Tableaux en Java

- **Un tableau est une séquence indexée d'éléments (valeurs) de même type**.
- Il est caractérisé par sa **taille** (nombre d'éléments qu'il peut contenir) et par le **type de ses éléments**.
- Le **type tableau** est un type référence (comme les objets).
- Les **éléments d'un tableau** peuvent être des valeurs de types primitifs, des objets ou des tableaux.

- Il faut bien faire la distinction entre les deux étapes :
  - la **déclaration du tableau** ([ ]) qui crée une variable (référence) représentant un objet de type tableau

  - et la **création du tableau** (new) qui alloue l'espace mémoire pour enregistrer les éléments du tableau et initialise le contenu



Déclaration de tableaux

Type_des_éléments [ ] *Identificateur* ;

- **Déclaration** de la variable identifiant l'objet de type tableau.
- La variable déclarée constitue une **référence** (sorte de pointeur vers une adresse mémoire) qui permet d'accéder au tableau.
- Le littéral null est une valeur particulière qui indique l'absence de référence (ou la référence vers rien).
- La taille du tableau n'est pas définie lors de sa déclaration mais seulement lors de sa création.

```
int[ ] intArray; // Déclaration d'un tableau de valeurs primitives int
Point[ ] nuage; // Déclaration d'un tableau d'objets Point
byte[ ][ ] matrix; // Déclaration d'un tableau de tableaux de byte
```

Déclaration de tableaux

Type_des_éléments **[ ]** *Identificateur* **=** *Initialisation* **;**

- Une **expression d'initialisation** peut être ajoutée à la déclaration; dans ce cas, le tableau est créé et la taille est déterminée par le résultat de l'expression d'initialisation.
- L'expression d'initialisation doit retourner un objet de type tableau compatible avec le type de la variable que l'on déclare.
- L'initialisation peut consister en une expression de création de tableau (avec l'opérateur new).
- Une syntaxe particulière permet de spécifier un **littéral tableau** comme expression d'initialisation : {*expr1*, *expr2*, *expr3*, ...}
  (cette syntaxe, sans new, n'est autorisée que lors de la déclaration)

  char[ ] vowels= {' a','e','i','o','u','y'};

Création de tableaux

```
new Type_des_éléments [ Taille ]
```

- La **taille du tableau** (nombre d'éléments) est déterminée par l'expression entre crochets qui doit avoir une valeur entière positive (ou zéro); la taille ne doit pas obligatoirement être connue à la compilation.
- Une fois qu'un tableau est créé, **sa taille ne peut plus varier**.
- Les **éléments du tableau** sont **initialisés à leurs valeurs par défaut** (false pour les booléens, 0 pour tous les autres types primitifs, null pour les objets et les tableaux).
- L'opérateur new **retourne une référence** à l'objet tableau créé.

```
int[ ] topTen = new int[10]; // Tableau de 10 entiers (int)
String[ ] text = new String[50]; // Déclaration et création simultanées
byte[ ][ ] matrix = new byte[25][80];
Point[ ] nuage = new Point[ astro.getDimension()];
```

Création de tableaux

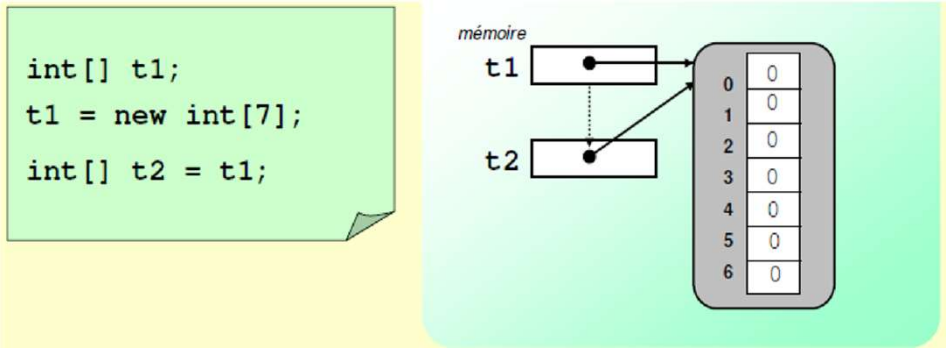
```
new Type_des_éléments [] {élém1, élém2, ...}
```

- **Combinaison de la création** d'un tableau **avec l'initialisation** de ses éléments (qui doivent être compatibles avec le type déclaré).
- La **taille du tableau** n'est pas spécifiée explicitement. Elle est déterminée par le **nombre d'éléments énumérés** entre les accolades (agrégat).

```
// Exemple de tableau crée et initialisé
  String []answers= new String[] {"Yes", "No"};
```

- Cette notation permet de créer des **tableaux anonymes** (non affectés à une variable) qui peuvent, par exemple, être passés en paramètre lors de l'invocation d'une méthode.

```
// Exemple de tableau anonyme passé à la méthode askQuestion
  String answer= askQuestion("Continue?", new String[] {"Yes", "No"});
```



Utilisation des tableaux

- Chaque **élément d'un tableau** est assimilé à une variable à laquelle on peut accéder individuellement. Par exemple : t[k]
- Chaque élément est identifié par un **indice entier** compris **entre 0 et (*n*-1), *n* étant la taille du tableau**
- **L'accès aux éléments d'un tableau s'effectue de la manière suivante :**

Nom_tableau [indice]

```
int[] tab; // Déclaration d'un tableau d'entiers (int)
tab = new int[3]; // Création du tableau avec 3 éléments
tab[0] = 7; // Assignation du premier élément
tab[1] = 4;
tab[2] = tab[0] + tab[1]; // Assignation et accès aux éléments
```

Utilisation des tableaux

- Le **nombre d'éléments d'un tableau** peut être obtenu à l'aide de l'attribut **length** qui est prédéfini pour tous les objets de type tableau (accessible en lecture uniquement).

  Exemple : nbElements = tab.length;

- Si, lors de l'accès à un élément d'un tableau, l'indice spécifié est négatif ou supérieur à l'indice du dernier élément (*length* -1), l'exception ***ArrayIndexOutOfBoundsException*** sera générée.
- L'expression qui détermine l'**indice** doit être de type byte, short, char ou int (le type long n'est pas autorisé).
- Le littéral null peut être utilisé pour représenter l'absence de tableau (comme pour tout autre objet)

  Exemple : char[] word = null;

Utilisation des tableaux

Tab1.java

```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```

Pour déclarer une variable tableau on indique le *type* des éléments du tableau et le *nom de la variable tableau* suivi de *[ ]*

on utilise new <type> [taille]; pour initialiser le tableau

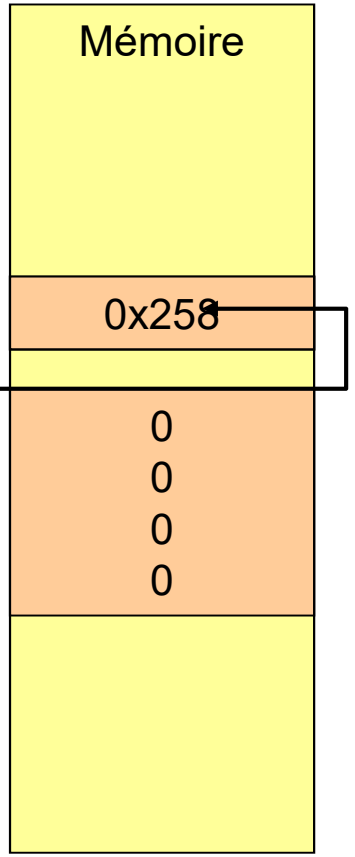
On peut ensuite affecter des valeurs au différentes cases du tableau : <nom_tableau>[indice]

Les indices vont toujours de 0 à (taille-1)

Les tableaux

Tab1.java

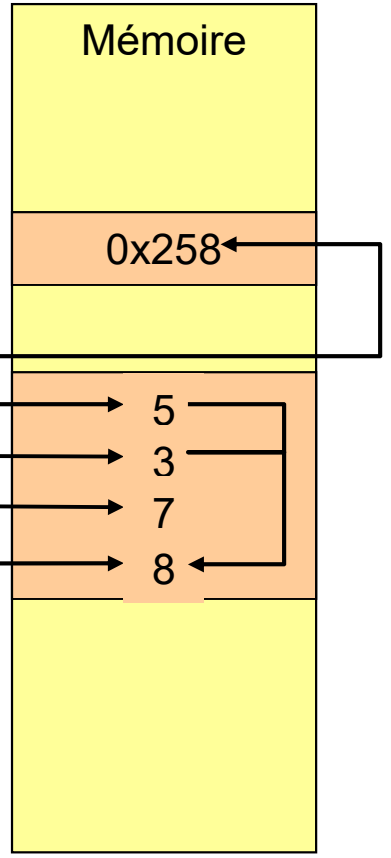
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



Les tableaux

Tab1.java

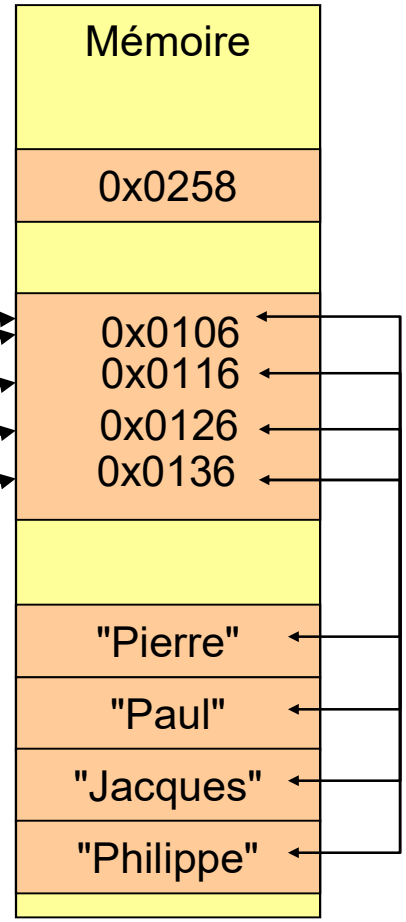
```
public class Tab1
{
  public static void main (String args[])
  {
    int tab[ ] ;
    tab = new int[4];
    tab[0]=5;
    tab[1]=3;
    tab[2]=7;
    tab[3]=tab[0]+tab[1];
  }
}
```



Les tableaux

Tab2.java

```
public class Tab2
{
  public static void main (String args[])
  {
    String tab[ ] ;
    tab = new String[4];
    tab[0]=new String("Pierre");
    tab[1]=new String("Paul");
    tab[2]=new String("Jacques");
    tab[3]=new String("Philippe");
  }
}
```



Accès aux éléments

```
public class TestArgs {
public static void main(String[] args) {
  System.out.println("nombre d ’arguments : " + args.length);
  for (int i =0; i < args.length; i++)
        System.out.println(" argument " + i + " = " + args[i]);}
  }
```

- **L’argument String[ ] args du main est un tableau de chaînes de caractères (String) correspondant aux arguments de la ligne de commande.**
- **Java TestArgs toto 23 titi**
- **Affiche sur la console les arguments**

nombre d ’arguments : 3

argument 0 = toto

argument 1 = 23

argument 2 = titi

Nouveauté à partir du JDK 1.5

- Une nouvelle syntaxe pour la boucle for➔For each

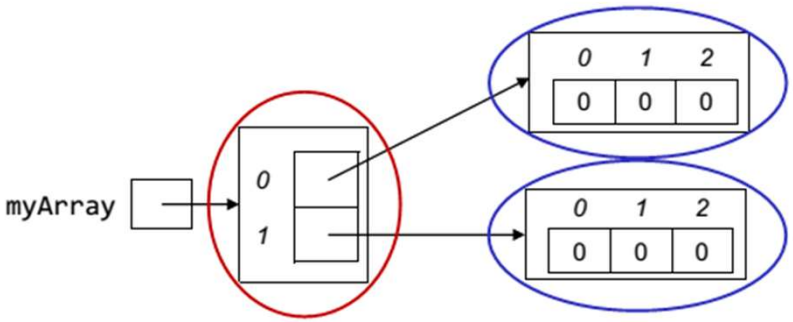
```
String tab[] = {"toto", "titi", "tutu", "tete", "tata"};
 for(String str : tab)
   System.out.println(str);
```

Attention cependant, il faut *impérativement* que la variable passée en premier paramètre de la boucle for soit de même type que la valeur de retour du tableau (une variable de type String pour un tableau de String, un int pour un tableau d'int etc.).

Tableaux Multidimensionnels

- Les tableaux peuvent avoir plusieurs dimensions. On parle alors de **tableaux multidimensionnels** (attention à ne pas confondre nombre de dimensions et taille du tableau).
- Un **tableau multidimensionnel** est un tableau de tableaux (de tableaux, de tableaux, ...).
- Chaque **paire de crochets** ([]) représente une **dimension** (aussi bien pour la déclaration, la création que l'accès aux éléments)

```
int[][] myArray = new int[2][3];    // Éléments initialisés à 0
```



Création

- **dimensions du tableau**
  - *ne sont pas spécifiées à la déclaration (comme pour les tableaux à une seule dimension).*
  - *indiquées que lors de la création*
    - *obligatoire que pour la première dimension.*
    - *autres dimensions peuvent n'être spécifiées que lors de la création effective des tableaux correspondants.*

```
double [][] matrice = new double[4][4];
```

Création d 'une matrice 4x4 de réels

```
double [][] matrice = new double[4][];
for (int i=0; i < 4; i++)
   matrice[i] = new double[4];
```

```
double [][] matrice;
matrice = new double[4][];
for (int i=0; i < 4; i++)
   matrice[i] = new double[4];
```

Les 3 écritures sont équivalentes

Tableaux multidimensionnels

Par contre, si l'on définit les tailles, il faut le faire dans l'ordre, **de gauche à droite**.

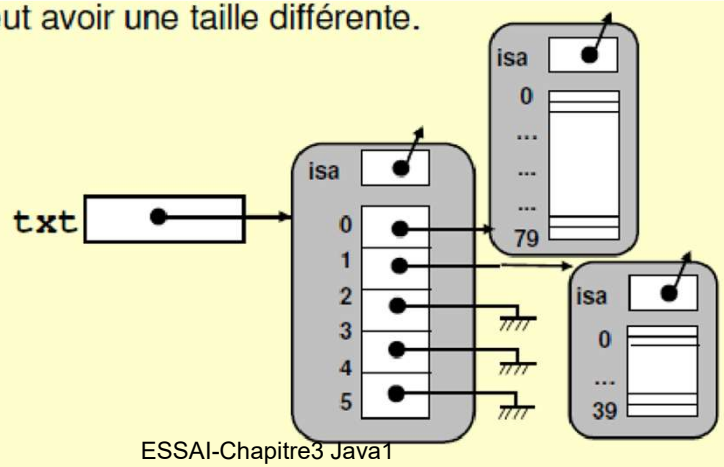
```
// Ok, déclaration de la première dimension
float[][][] colorPalette = new float[10][][];

// Ok, déclaration des deux premières dimensions
float[][][] colorPalette = new float[10][20][];

// Erreur à la compilation !
float[][][] colorPalette = new float[][20][];
```

- chaque tableau imbriqué peut avoir une taille différente.

```
char [][] txt;
txt = new char[6][];
txt[0] = new char[80];
txt[1] = new char[40];
txt[2] = new char[70];
...
```



Accès aux éléments

- **accès aux éléments d'un tableau multidimensionnel (exemple 2d)**
  - *accès à un élément d'un tableau s'effectue à l'aide d'une expression de la forme :*

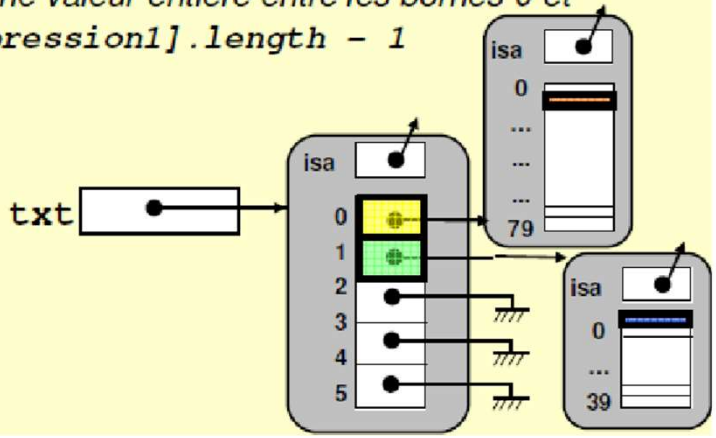
    nomDuTableau[expression1][expression2]

    où

    – *expression1 délivre une valeur entière entre les bornes 0 et* ***nomDuTableau.length - 1***

    – *expression2 délivre une valeur entière entre les bornes 0 et* ***nomDuTableau[expression1].length - 1***

```
char[][] txt = new char[5][];
txt[0] = new char[80];
txt[1] = new char[40];
txt[0][1] = 'S';
txt[1][0] = 'H';
```



Création

- **Comme pour tableaux unidimensionnels possible de créer un tableau multidimensionnel en donnant explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades)**
  - *exemples :*

```
int[][] t1 = {
        { 1, 2, 3, 4, 5},
        { 6, 7, 8, 9, 10},
    };
```

```
int[][] t1 = new int[2][5];
t1[0][0] = 1; t1[0][1] = 2; ...
t1[1][0] = 6; t1[1][1] = 7;...
```

Cette virgule finale n'est pas une faute de frappe, elle est optionnelle et est juste là pour permettre une maintenance plus facile de longues listes (et faciliter les couper/ coller des programmeurs pressés… :-)

Les **tableaux multidimensionnels** étant des tableaux de tableaux, ils ne doivent **pas obligatoirement être rectangulaires** (ou cubiques ou ...).

```
//-------------------------------------------------------------
// Tableau de 7 éléments dont chacun représente
// un tableau de taille variable
// (tableau triangulaire)
//-------------------------------------------------------------

int[][] multTable = { {0},
                      {0,  1},
                      {0,  2,  4},
                      {0,  3,  6,  9},
                      {0,  4,  8, 12, 16},
                      {0,  5, 10, 15, 20, 25},
                      {0,  6, 12, 18, 24, 30, 36} };
```

For each

```
String tab[][]={{"toto", "titi", "tutu", "tete", "tata"}, {"1", "2", "3", "4"}};
int i = 0, j = 0;

for(String sousTab[] : tab)
{
  i = 0;
  for(String str : sousTab)
  {
    System.out.println("La valeur est  : " + str);
    System.out.println("La valeur du tableau à l'indice ["+j+"]["+i+"] est : " +
  tab[j][i]);
    i++;
  }
  j++;
}
```

Tableaux
à propos de la classe Arrays

- package `java.util` définit une classe, **`Arrays`**, qui propose des méthodes statiques (de classe) pour le tri et la recherche dans des tableaux.

Exemple : tri d'un tableau

```
// tableau de 1000 réels tirés au hasard
// dans l 'intervalle [0..1000[
double[] vec = new double[1000];
for (int i = 0; i < vec.length; i++)
      vec[i] = Math.random()*1000;

// tri du tableau
Arrays.sort(vec);

// affiche le tableau trié
for (int i = 0; i < vec.length; i++)
      System.out.print(vec[i] + " " );
```

Le tri.
Dans l 'implémentation de SUN une variation du QuickSort
(O(n*log(n))

Tableaux
à propos de la classe Arrays

- **pour chaque type de tableau**
  - *Des méthodes de recherche*
    - *int binarySearch(char[ ] a) , int binarySearch(int[ ] a) ….*

    … int binarySearch(Object[ ] a)
  - *Des méthodes de tris*
    - *sort(char[ ] a) , sort(int[ ] a) ….. sort(Object[ ] a)*
    - *sort(char[ ] a, int fromIndex, int toIndex) , …*
  - *Des méthodes pour remplissage avec une valeur*
    - *fill(char[ ] a, char val) , fill(int[ ] a, long val) …..*
    - *fill(char[ ] a, char val, int fromIndex, int toIndex) , …*
  - *Des méthodes de test d'égalité*
    - *boolean equals(char[ ] a1, char[ ] a2), boolean equals(int[ ] a1, int[ ] a2), …..*

Tableaux
à propos de java.util

- Les tableaux sont des structures de données élémentaires
- Le package **java.util** contient plein de classes « sympa » pour la gestion de structures de données plus évoluées (collections) :
  - *listes*
  - *ensembles*
  - *arbres*

mais pour apprécier il faudra être un peu patients…

parlons d'abord d'objets, d'héritage, de classes abstraites et d'interfaces !

```
public class ExempleTableau {
 public static void main(String[] args) {
    // DECLARATION
    double[] tab;

    // CREATION ET DIMENSIONNEMENT
    int dim = 1+(int)(9*Math.random());
    tab = new double[dim];

    // AFFECTATION DES ELEMENTS
    //              DU TABLEAU
    for (int i=0; i<tab.length; i++)
      tab[i] = 1./(i+1);

    // AFFICHAGE
    for (int i=0; i<tab.length; i++)
      System.out.print(tab[i]+" ");
    System.out.println();
 }
}
```

Dimension du tableau

Copie de tableau

- Méthode « manuelle » : création d'un nouveau tableau puis copie case par case
- Copie par appel de System.arraycopy() :

```
// copie des 50 premiers éléments de src dans dst à partir de l'indice 2
System.arraycopy(src, 0, dst, 2, 50);
```

- Allocation et copie par appel de Arrays.copyOf() ou Arrays.copyOfRange() :

```
// copie des 10 premiers éléments :
int[] t2= Arrays.copyOf(t, 10);
// copie entre les indices 30 à 50 :
int[] t3= Arrays.copyOfRange(t, 30, 50);
```

- Allocation et copie par clonage :

```
int[] copie = t.clone();
```

- **Attention : pour un tableau d'*objets, copie des*** références => les mêmes objets sont ensuite partagés par les 2 tableaux
- Note : déplacement interne dans un tableau possible par System.arraycopy() :

```
// par ex. décalage de 1 vers la droite:
```

- System.arraycopy(t,0,t,1,t.length-1);

Référence/objet référencé

Avec les types référence, il faut toujours **bien distinguer** si l'on traite la **référence** (*adresse*) ou l'**objet référencé** (*contenu*). Cette distinction est **très importante**, notamment avec les opérateurs d'égalité (==), d'inégalité (!=) et d'affectation (**=**) ainsi que lors des passages en paramètre (invocation de méthodes).

```
char[] c1 = {'a', 'b', 'c'};
char[] c2 = {'a', 'b', 'c'};
char[] c3 = c1;

if (c1 == c2) System.out.println("c1 égal à c2");
if (c1 == c3) System.out.println("c1 égal à c3");
if (c2 == c3) System.out.println("c2 égal à c3");

if (c1[0] == c2[0]) System.out.println("c1[0] == c2[0]");
if (c1[0] == c3[0]) System.out.println("c1[0] == c3[0]");
if (c2[0] == c3[0]) System.out.println("c2[0] == c3[0]");

c3[0] = 'z';
if (c1[0] == c3[0]) System.out.println("c1[0] == c3[0]");
if (c2[0] == c3[0]) System.out.println("c2[0] == c3[0]");
```