

# ***Tableaux en Java***

- Les tableaux = **objets** en java (et pas seulement une suite d'emplacements mémoire comme en C/C++)
- Fournissent des collections ordonnées d'éléments
- Composants d'un tableau peuvent être :
  - *des variables des types de base (int, boolean, double, char...)*
  - *des références sur des objets,*
    - *y compris des références sur d'autres tableaux (les tableaux sont effectivement des objets en JAVA)*

# Tableaux

## Tableaux unidimensionnels

- **Déclaration**

- *typeDesElements[] nomDuTableau;*

*ou bien*

- *typeDesElements nomDuTableau[];*

*avec*

- *typeDesElements* un des types de base du langage JAVA (*char, int, float, double...*) ou un nom de classe
    - *nomDuTableau* l'identificateur pour désigner le tableau

(des deux formes de déclaration on préférera la première car elle place la déclaration de type en un seul endroit).

- **exemples**

```
int vecteurEntiers[];  
int[] vecteurEntiers; // identique  
  
Compte[] listeDesComptes;
```

## Tableaux unidimensionnels

- la taille d'un tableau n'est pas spécifiée à la déclaration.
- les tableaux sont des objets,
  - `typeDesElements[] nomDuTableau;`
  - *défini une **référence** vers un objet de type "tableau d'éléments de type typeDesElements".*
  - *le nombre d'éléments du tableau sera déterminé quand l'objet tableau sera effectivement créé en utilisant `new`.*
  - *cette taille fixée à la création est fixe, elle ne pourra plus être modifiée par la suite.*
- exemple

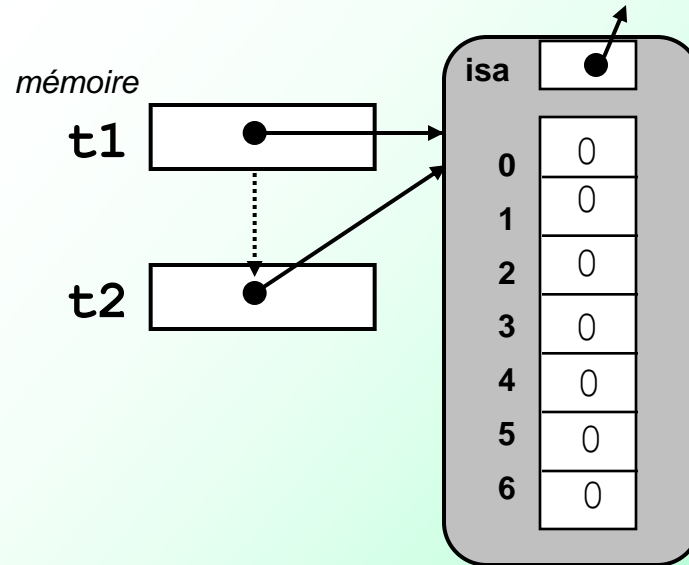
```
int[] vecteurEntiers;  
vecteurEntiers = new int[50];  
  
Compte[] listeDesComptes = new Compte[1000];
```

- La création d'un tableau par **new**
  - *alloue la mémoire nécessaire en fonction*
    - *du type du tableau*
    - *de la taille spécifiée*
  - *initialise le contenu du tableau*
    - *type simple : 0*
    - *type complexe (classe) : **null***
      - *en effet quand on déclare un tableau d'un type d'objets, on déclare un tableau de **références** de ce type.*
      - *A la création du tableau ces références sont toutes initialisées à **null** (elles ne pointent vers aucun objet).*
      - *Les objets doivent être ensuite créés explicitement et "rangés" dans le tableau.*

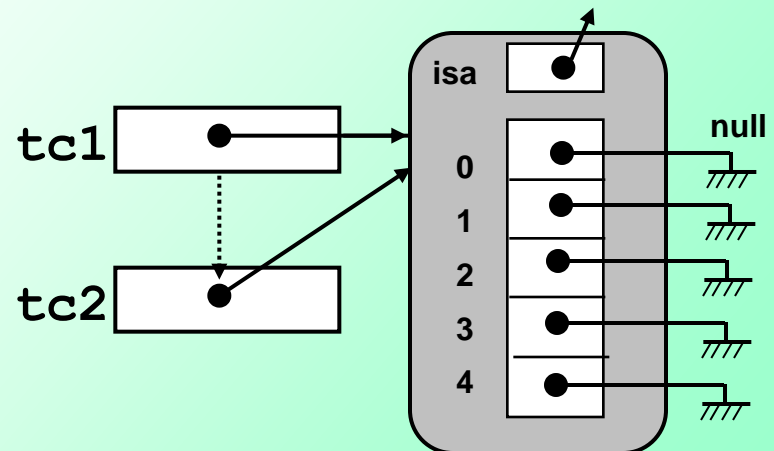
# Tableaux

## Tableaux unidimensionnels

```
int[] t1;  
t1 = new int[7];  
int[] t2 = t1;
```



```
Compte[] tc1;  
tc1 = new Compte[5];  
Compte[] tc2 = tc1;
```



## Tableaux unidimensionnels

- accès aux éléments d'un tableau unidimensionnel

- *comme en C/C++ indices commencent à zéro*
- **length** donne la dimension (entier)
  - *tabEntiers.length-1 = indice max de tabEntiers*
- *accès à un élément d'un tableau s'effectue à l'aide d'une expression de la forme :*

***nomDuTableau[expression1]***

*où*

- *expression1 délivre une valeur entière dans les bornes du tableau (>= 0 et <length)*
- *on peut rencontrer une telle expression aussi bien en partie gauche qu'en partie droite d'affectation*
- *Java vérifie automatiquement l'indice lors de l'accès (comparaison avec la borne)*
  - *Si hors limites : `ArrayIndexOutOfBoundsException`*
  - *Evite des bugs ! Mais cela a un coût ...*

## Tableaux unidimensionnels

- à propos de `length`

- `length` donne la dimension (entier)*

- `tabEntiers.length-1` = indice max de `tabEntiers`*

```
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]);  
    }  
}
```

- `length` est une variable d'instance publique**

- Et le principe d'encapsulation alors ?!!?*

- `length` est déclarée `public final`**

- Elle ne peut être modifiée après son initialisation à la création du tableau*

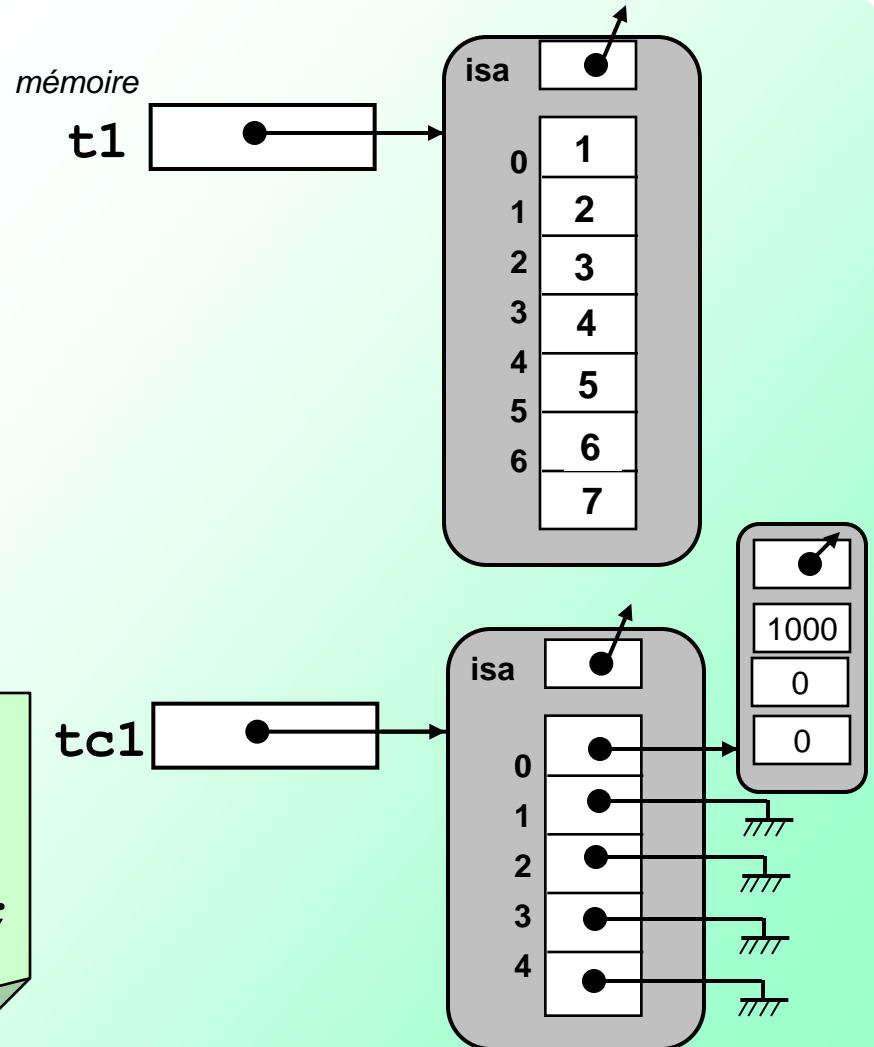


# Tableaux

## Tableaux unidimensionnels

```
int[] t1;  
t1 = new int[7];  
  
t1[0] = 1;  
for (int i=1; i<7; i++)  
    t1[i] = t1[i-1]+1;
```

```
Compte[] tc1;  
tc1 = new Compte[5];  
tc1[0] = new Compte(1000);
```



- Une autre manière de créer des tableaux :

- *en donnant explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades)*
- *exemples :*

```
int[] t1 = { 1, 2 ,3, 4, 5};
```

```
Compte[] cpts = { new Compte(2000) ,  
                  new Compte(2001) , new Compte(2002) };
```

- *l'allocation mémoire (équivalent de l'utilisation de **new**) est prise en charge par le compilateur*

## Tableaux multidimensionnels

- tableau dont les éléments sont eux mêmes des tableaux
- un tableau à deux dimensions se déclarera ainsi de la manière suivante :

```
typeDesElements[][] nomduTableau;
```

- *exemples*

- `double[][] matrice;`
- `Voxel[][][] cubeVoxels;`

## Tableaux multidimensionnels

- **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];
```

Les 3 écritures sont  
équivalentes

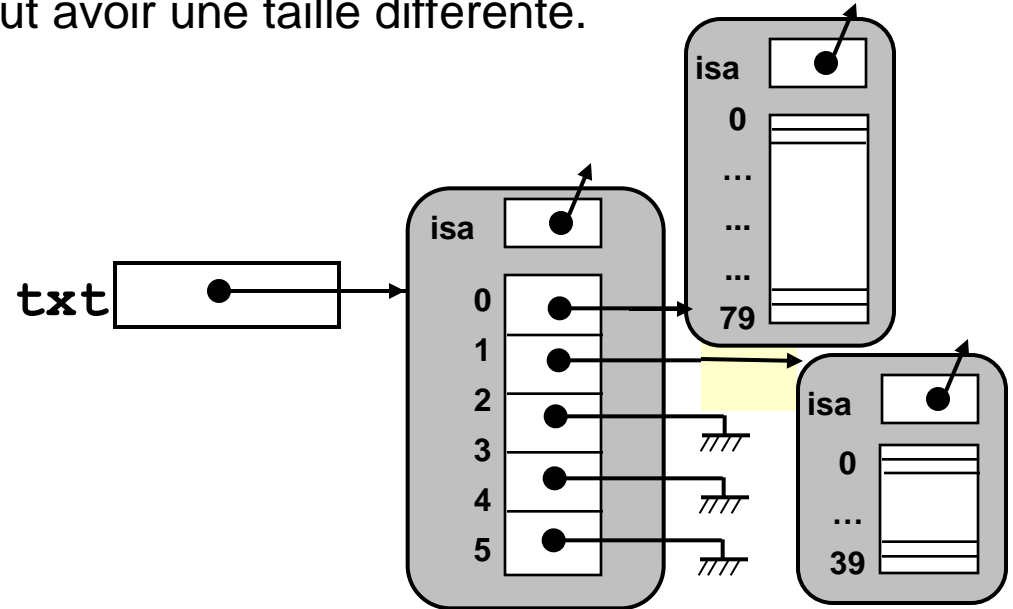
```
double [][] matrice;  
matrice = new double[4][];  
for (int i=0; i < 4; i++)  
    matrice[i] = new double[4];
```

# Tableaux

## Tableaux multidimensionnels

- 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];  
...
```



# Tableaux

## Tableaux multidimensionnels

- accès aux éléments d'un tableau multidimensionnel (exemple 2d)

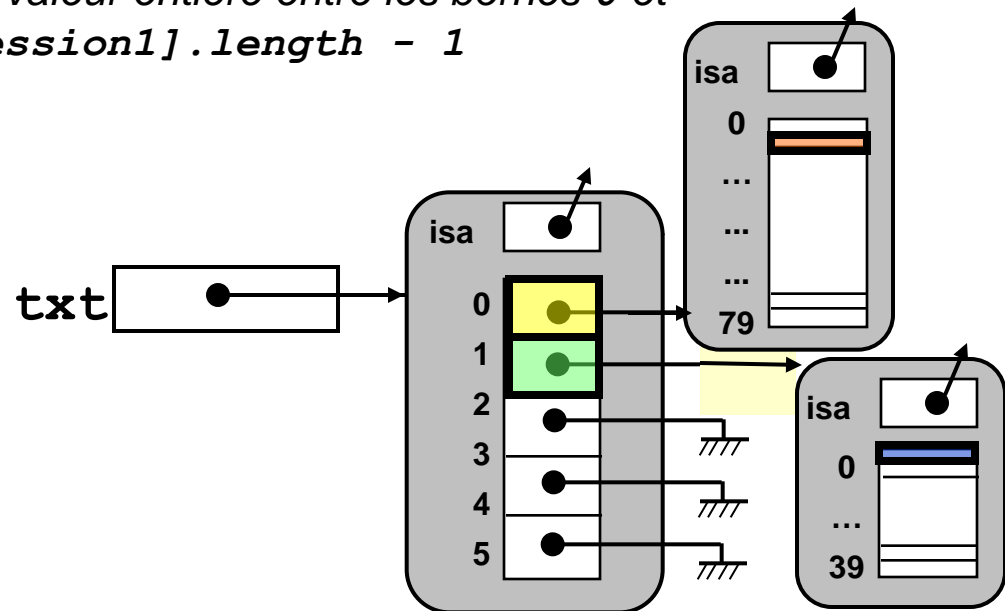
- accès à un élément d'un tableau s'effectue à l'aide d'une expression de la forme :

***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';
```



## Tableaux multidimensionnels

- 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;...
```

```
Point[][] pts = {  
    { new Point(15,11), new Point(70,14) },  
    { new Point(3,61), new Point(10,6) },  
    { new Point(57,22), new Point(8,93) }  
};
```

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

# 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  
( $O(n \cdot \log(n))$ )



# Tableaux

## à propos de la classe Arrays

Exemple : recherche dans un tableau

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

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

// recherche de la valeur 500
int pos = Arrays.binarySearch(vec, 500);

// utilisation des résultats de la recherche
if (pos >= 0)
    System.out.println("position de 500 : " + pos);
else {
    System.out.println("500 n'est pas dans le tableau");
    System.out.println("position d'insertion : " + -(pos+1));
}
```

Il faut que le tableau soit trié avant toute recherche

La recherche

## 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), .....*

- boucle foreach applicable aussi sur des tableaux d'objets

```
/**
 * translate tous les points contenus dans un tableau
 * @param pts le tableau de points à traduire
 * @param x abscisse du vecteur de translation
 * @param y ordonnée du vecteur de translation
 */
public void traduire(Point[] pts, double x, double y) {

    for (int i = 0; i < pts.length; i++)
        pts[i].traduire(x,y);

}
```

Boucle for traditionnelle

Avec le nouveau  
type de boucle for

```
public void traduire(Point[] pts,
                    double x, double y) {

    for (Point p : pts)
        p.traduire(x,y);

}
```

Lire : pour  
chaque Point  
p de pts

- Boucle « *foreach* » introduite depuis la version 1.5 de Java peut être utilisée pour itérer sur des tableaux

```
/**
 * calcule somme des éléments d'un tableau d'entiers
 * @param a le tableau d'entier
 * @return la somme des éléments de a
 */
public int sum(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++)
        sum += a[i];
    return sum;
}
```

Boucle for traditionnelle

```
public int sum(int[] a) {
    int sum = 0;
    for (int x : a)
        sum += x;
    return sum;
}
```

Avec le nouveau  
type de boucle for

Lire : pour  
chaque x de a

# 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 'héritage,
  - de classes abstraites
  - et d 'interfaces ;-) !