Java
Les Tableaux

Ahcène Bounceur

université de bretagne occidentale

Les tableaux en java

- Les tableaux en java sont des objets !!
- Un tableau mutldimensionnel est une composition de plusieurs tableaux
- Déclaration d'un tableau d'entiers :

```
int [] t ;
```

Les tableaux en java

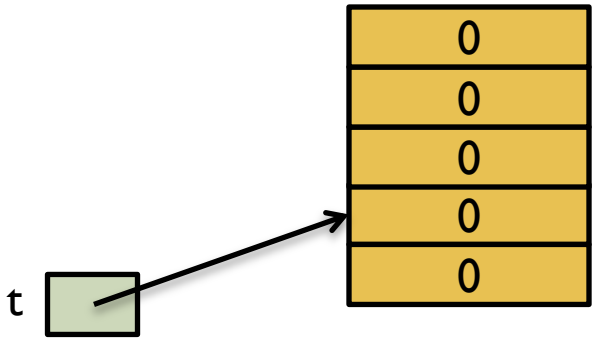
- On peut aussi écrire :

```
int t [] ;
```

- C'est juste une déclaration d'un tableau. Il n'y au aucune allocation mémoire. C'est-à-dire que le tableau n'est pas encore instancié !!
- Donc : t est une référence
- Autres manières de déclaration de tableaux :
  - int t1 [], n, t2[] ;
  - Point p [], c ;

Les tableaux en java

- Pour créer un tableau, il faut donc l'instancier comme on instancie un objet :
  - **int [] t = new int [5] ;**



Les tableaux en java

- Attention ! La déclaration suivante est fausse

  int t [5] ;

- Initialisation :

  int t [] = { 1, n, n+p, 2*p, 12 } ;

  Cette initialisation remplace :

  ```
  int [] t = new int [5] ;
  t[0] = 1 ;
  t[1] = n ;
  t[2] = n+p ;
  t[3] = 2*p ;
  t[4] = 12 ;
  ```

Les tableaux en java

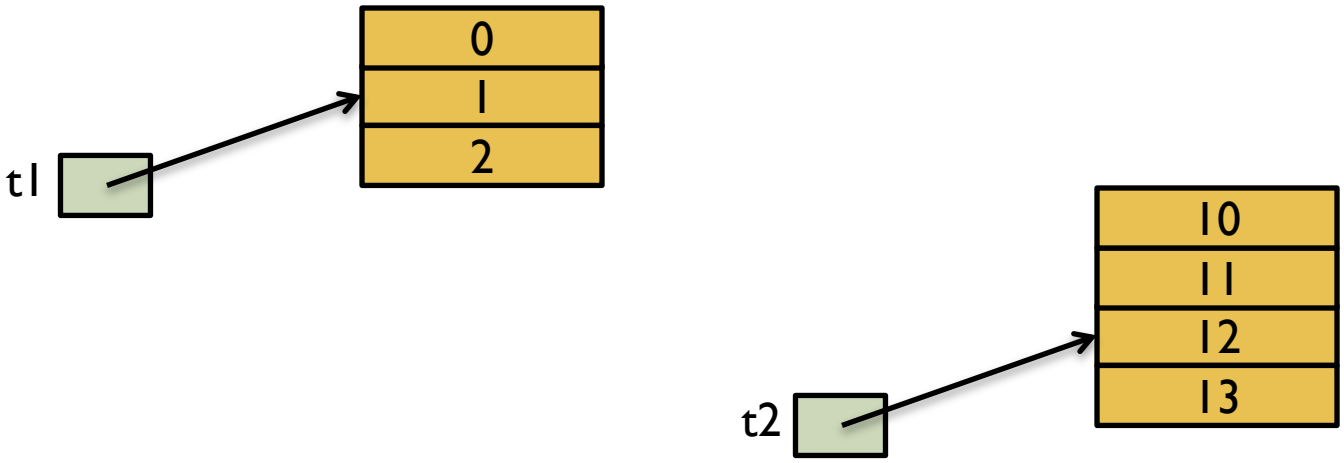
- Utilisation des tableaux :

```
int [] t = new int [5] ;
t[0] = 15 ;
t[2]++ ;
System.out.println("x = "+t[4] );
```

Affectation de tableaux

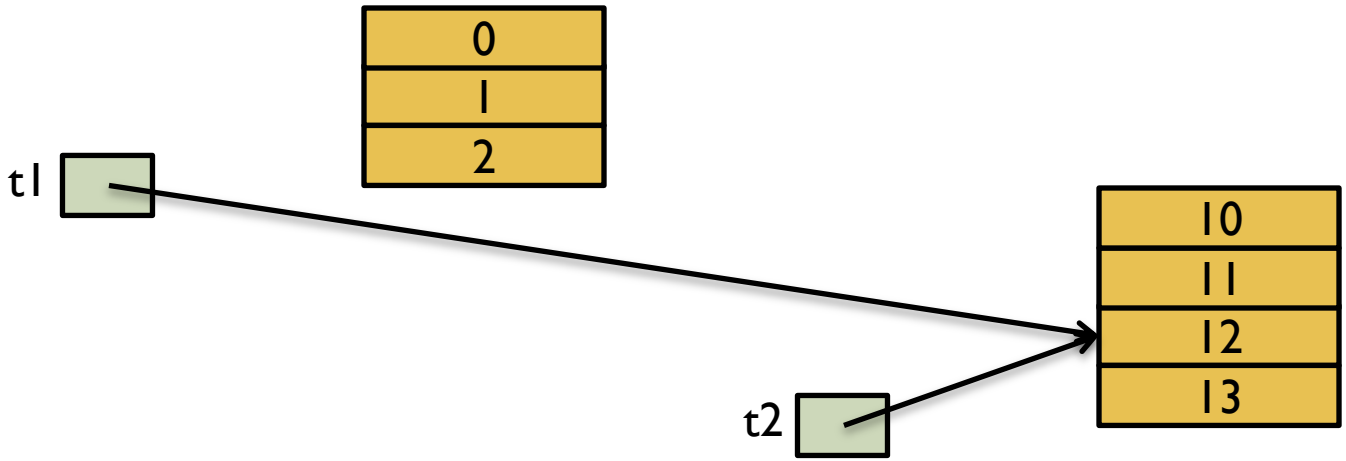
```
int [] t1 = new int [3] ;
for (int i = 0; i < 3; i++) t1[i] = i ;

int [] t2 = new int [4] ;
for (int i = 0; i < 4; i++) t2[i] = 10+i ;
```



Affectation de tableaux

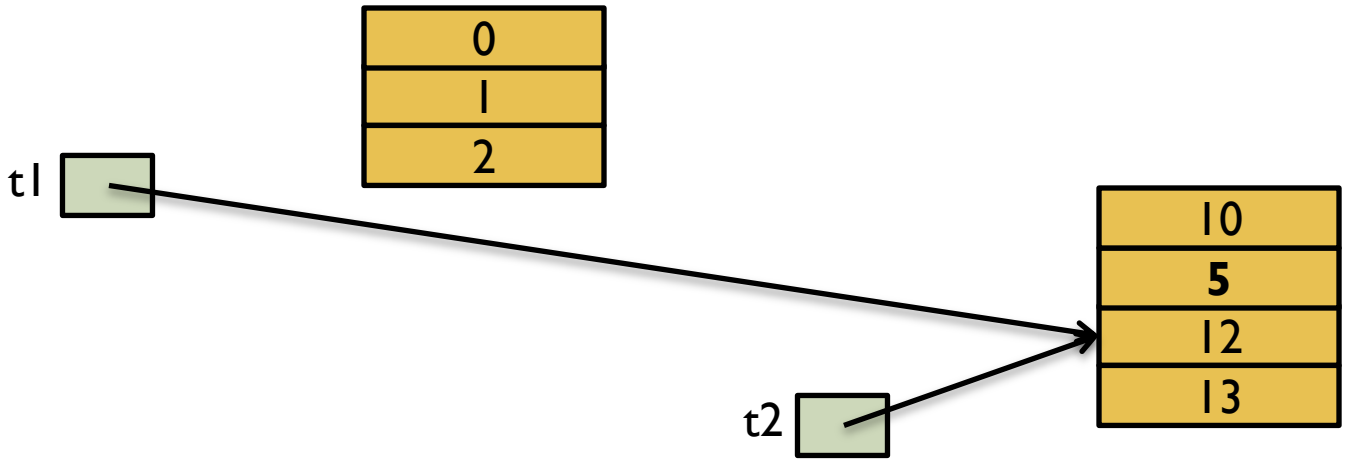
```
t1 = t2 ;
```



Affectation de tableaux

```
t1[1] = 5 ;
System.out.println(t2[1]) ;
```

```
Résultat :
5
```



La taille d'un tableau : **length**

```
int [] t = new int [3] ;
System.out.println("taille de t : "+ t.length ) ;
```

```
Résultat :
3
```

- Attention !
  - On écrit **length** et non pas **length()**

Tableau d'objets

```
Point [] tp = new Point [3] ;
tp[0] = new Point(1, 2) ;
tp[1] = new Point(7, 3) ;
tp[2] = new Point(4, 5) ;
for (int i = 0; i < 3; i++)
        tp[i].afficher() ;
```

La boucle **for … each**

```
// de manière classique

double [] t = new double [20] ;
for (int i = 0; i < t.length; i++)
        System.out.println(t[i]) ;
```

```
// utilisation de for ... each

double [] t = new double [20] ;
for (double v : t)
        System.out.println(v) ;
```

- **La boucle for … each sert surtout pour accéder aux valeurs d'un tableau et non pas pour faire des affectations !**

Tableau en argument

- Un tableau passé en argument, c'est sa référence qui est copiée. Donc, toute modification de l'argument entrainera la modification du tableau envoyé.

Tableau multidimensionnel

- Ou tableau à plusieurs indices
- Déclarations d'un tableau à 2 dimensions :

```
int t [] [] ;
int [] t [] ;
int [] [] t ;
```

Tableau multidimensionnel

- Initialisation classique :

```
int [][] t = new int [3][2] ;
for (int i = 0; i < t.length; i++) {
        for (int j = 0; j < t[i].length; j++) {
                t[i][j] = 0 ;
}
```

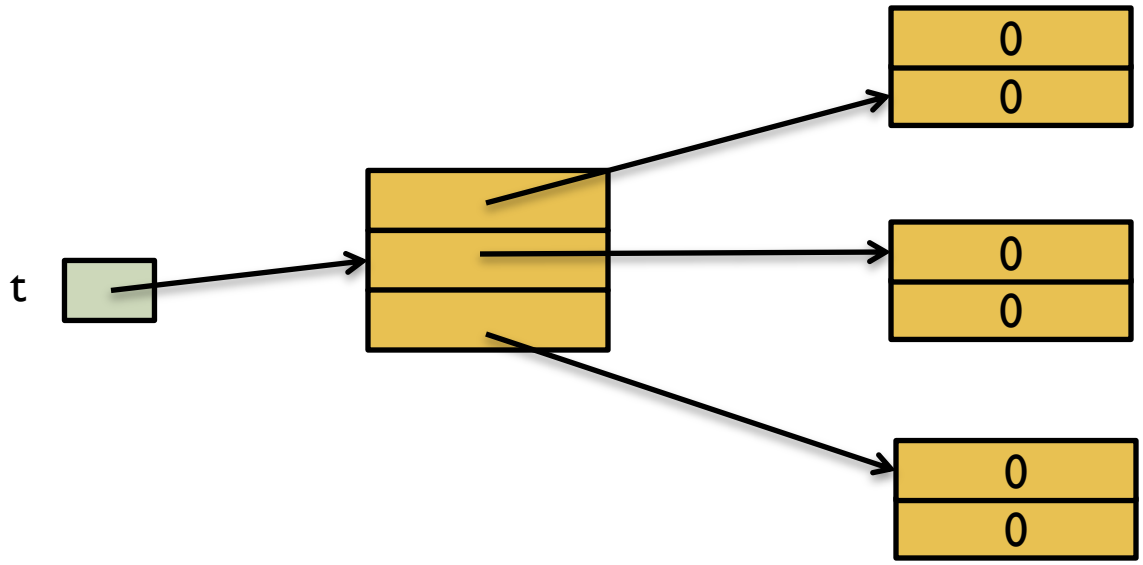
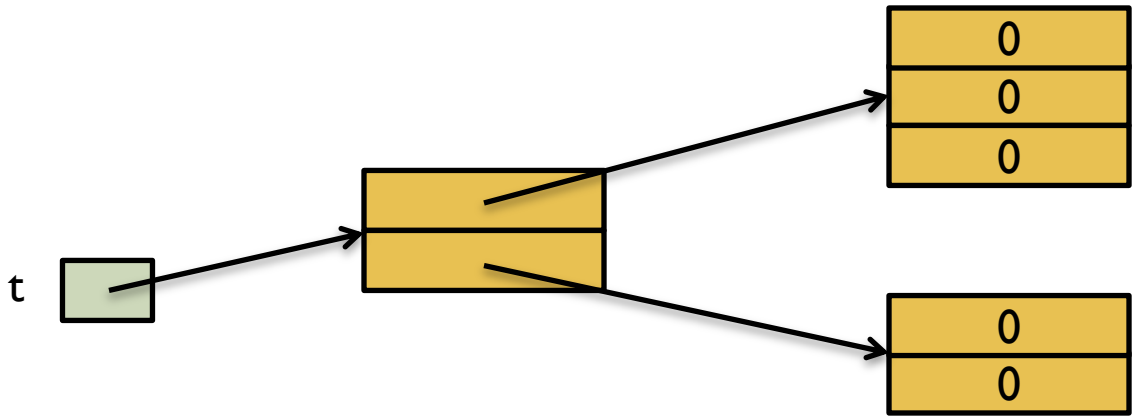


Tableau multidimensionnel

- Initialisation :

```
int [][] t = { new int[3],  new int[2] } ;
```



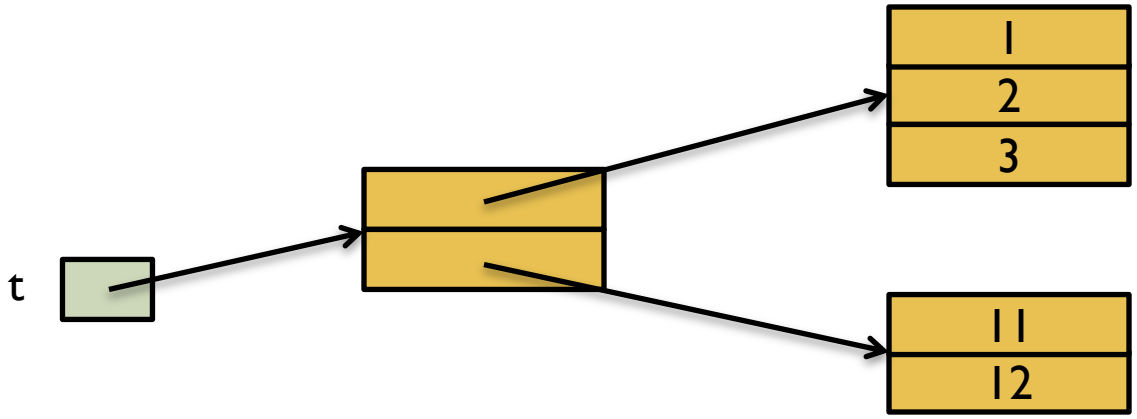
Remarque :

`t[0].length` **vaut** `3`

Tableau multidimensionnel

- Initialisation :

```
int [][] t = { {1,2,3}, {11,12} } ;
```



- Un autre exemple. Il sert à initiliser dynamisuqemnet la deuxième dimension.

```
int [][] t = new int [2][] ;
```

- Par exemple : t[0] = new int [5] ;

Tableau multidimensionnel : exemple

```
public class TestTableau {
    public static void main (String [] args ) {
        int [][] t = new int [3][] ;
        t[0] = new int [5] ;
        t[1] = new int [8] ;
        t[2] = new int [3] ;
        int c = 0 ;
        for (int i = 0; i < t.length; i++) {
            for (int j = 0; j < t[i].length; j++) {
                t[i][j] = c++ ;
            }
        }
        for (int i = 0; i < t.length; i++) {
            for (int j = 0; j < t[i].length; j++) {
                System.out.print(t[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

Résultat :

```
0 1 2 3 4
5 6 7 8 9 10 11 12
13 14 15
```

Tableau multidimensionnel : **for … each**

```
int [][] t = new int [3][2] ;

for (int [] ligne : t) {
        for (int v : ligne)
                System.out.print(v + " ") ;
        System.out.println() ;
}
```

Un nombre d’arguments variable !

- Depuis le JDK 5.0 on peut définir une méthode dont le nombre d'arguments est variable !!!

Un nombre d'arguments variable !

- Déclaration (exemple) :

```
public static int somme(int ... valeurs) {
   ...
}
```

- L'argument **valeurs** sera manipulé comme étant un tableau. C'est comme si l'on avait en argument **int [] valeurs**

Un nombre d'arguments variable !

- Exemple :

```
public class Arguments {
    public static int somme(int ... valeurs) {
        int s = 0 ;
        for (int i = 0; i < valeurs.length; i++) {
            s += valeurs[i] ;
        }
        return s;
    }
}
```

On appelle ces trois points :
ELLIPSE

```
public class Test {
    public static void main(String [] args) {
        System.out.println(Arguments.somme());
        System.out.println(Arguments.somme(5,4));
        System.out.println(Arguments.somme(2,6,3));
    }
}
```

```
Résultat :
0
9
11
```

Un nombre d'arguments variable !

- Exemple : utilisation du for … each

```
public class Arguments {
    public static int somme(int ... valeurs) {
        int s = 0 ;
        for (int v : valeurs) {
            s += v ;
        }
        return s;
    }
}
```

```
public class Test {
    public static void main(String [] args) {
        System.out.println(Arguments.somme());
        System.out.println(Arguments.somme(5,4));
        System.out.println(Arguments.somme(2,6,3));
    }
}
```

```
Résultat :
0
9
11
```

Un nombre d'arguments variable !

- Remarques :
  - Dans le cas d'une méthode à plusieurs arguments et si parmi eux il y a une ellipse, celle-ci doit être mise en dernier dans la liste

```
public int somme(int n, int v, int ... valeurs) {...}
```

```
public int somme(int ... valeurs, int n, int v) {...}// erreur
```

```
public int somme(int n, int ... Valeurs, int v) {...}// erreur
```

  - Ne pas mettre deux ellipses dans les arguments d'une méthode :

```
public int somme(int ... val1, int ... val2) {...}// erreur
```

Un nombre d'arguments variable !

- Remarques :
  - Surdéfinir deux méthodes, une avec un seul argument représentant une ellipse et l'autre avec un seul argument représentant un tableau est incorrect

```
public int somme(int ... Valeurs) {...}
public int somme(int [] Valeurs) {...}
```

  - La surdéfinition suivante est correcte : la méthode (2) est prioritaire

```
public int somme(int ... Valeurs) {...}      // (1)
public int somme(int a, int b) {...}         // (2)
```

- Merci de votre attention