

Cours de Programmation Orientée Objet JAVA

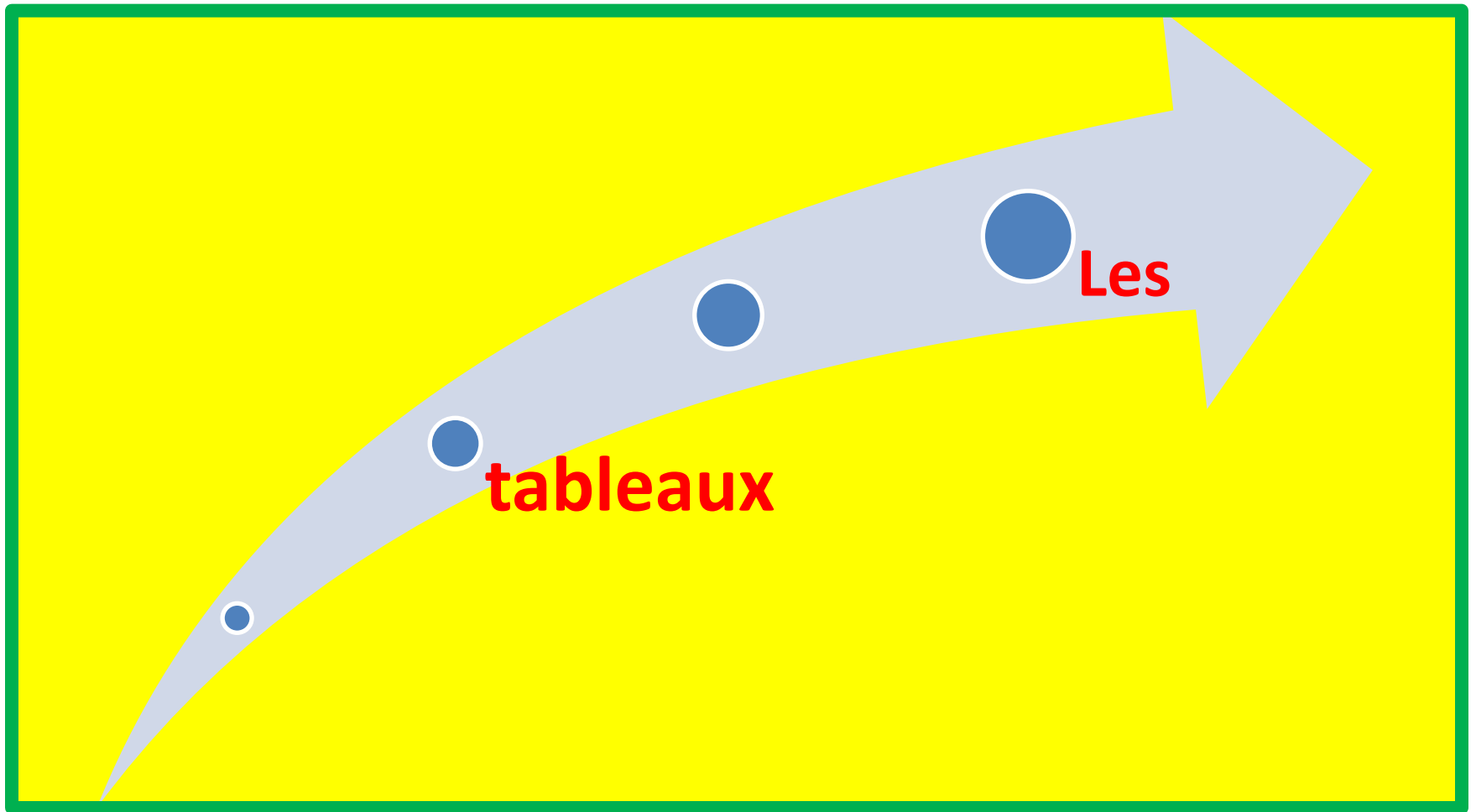
Demba SOW

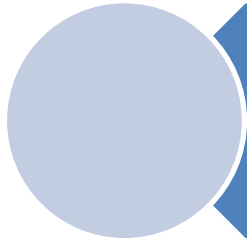
*Docteur en Codage
Cryptologie Algèbre et
Applications*

L.A.C.G.A.A.

F.S.T. / U.C.A.D.







Les tableaux

Introduction

Les tableaux sont des structures de données regroupant plusieurs valeurs de même type.

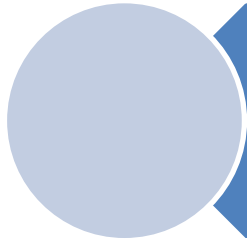
Ou encore on parle de tableaux pour désigner un ensemble d'éléments de même type désignés par un nom unique, chaque élément étant repéré par un indice précisant sa position au sein de l'ensemble .

Les tableaux constituent des **collections d'informations homogènes**, c'est-à-dire, de valeurs primitives ou d'objets de même type.

Les éléments d'un tableau peuvent être :

- des primitives (scalaires) (float, int, char, etc.),
- des références d'objets (String, Object),
- des références de tableaux.

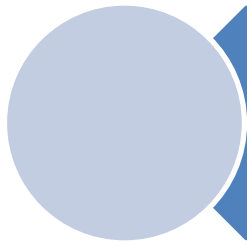
La taille d'un tableau est fixée d'une façon permanente suite à la déclaration du tableau et à l'allocation de ressources systèmes pour ce dernier.



Introduction

La *taille* d'un tableau est donc **fixée lors de sa création** et ne peut être plus être changée pendant toute la durée de sa vie.

Une solution est de créer un tableau d'une taille donnée, et, lorsque celui-ci est saturé, en créer un nouveau et déplacer toutes les références de l'ancien tableau dans le nouveau. C'est précisément ce que fait la classe **ArrayList** ou la classe **Vector**, qui seront étudiées plus loin dans ce cours.



Introduction

Les tableaux sont des objets:

- Indépendamment du type de tableau qu'on utilise, un identifiant de tableau est en fait une référence sur un vrai objet créé dans le segment.
- C'est l'objet qui stocke les références sur les autres objets, et il peut être créé soit implicitement grâce à la syntaxe d'initialisation de tableau, soit explicitement avec une expression **new**.
- **length** indique combien d'éléments peuvent être stockés dans l'objet. La syntaxe « **[]** » est le seul autre accès disponible pour les objets tableaux.



Déclaration et création de tableaux

Déclaration.

type identificateur [] ; // on peut déclarer un tableau comme ceci

type [] identificateur ; // ou comme cela

Exemples:

int t [] ; // *t est destiné à contenir la référence à un tableau d'entiers.*

// on peut aussi écrire **int [] t**

Object [] obj ; // *obj est destiné à contenir la référence à un tableau d'objets*

En fait la différence entre les deux formes de déclaration devient perceptible lorsque l'on déclare plusieurs identificateurs dans une même instruction .

Ainsi :

```
int [ ] t1 ,t2 ; // t1 et t2 sont des références à des tableaux d'entiers
```

```
int t1 [ ], n, t2 [ ] ; // t1 et t2 sont de tableaux d'entiers , n est entier
```



La taille d'un tableau n'est spécifiée qu'à partir du moment de son utilisation dans le programme. Ainsi, la mémoire ne sera allouée que lorsque cela sera nécessaire.

C'est pourquoi, lors de la déclaration du tableau vous ne pouvez pas faire:

```
int t [12]; // NON, pas de dimension.
```



Création.

On crée un tableau comme on crée un objet, c'est-à-dire en utilisant l'opérateur **new**. On précise à la fois le type des éléments, ainsi que leur nombre (dimension ou taille du tableau).

En d'autres termes la définition d'une référence d'un tableau, c'est-à-dire la spécification de la taille du tableau référencé par la variable, s'accomplit comme ceci:

identificateur = new type[taille]; //le système alloue un emplacement
//mémoire pour un tableau de taille éléments de type type .

Exemples:

t = new int [10]; // la variable t fait référence à un tableau de 10
// valeurs entières.

La déclaration peut se combiner à la définition du tableau produisant une instruction plus compacte.

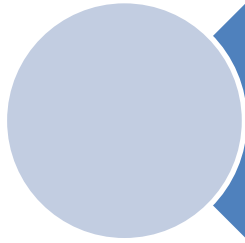
Dimension obligatoire

```
String [ ] tabcar = new String [ 14];
```

Dans la création d'un tableau, il faut obligatoirement mentionner la taille du tableau.

Par défaut, les valeurs de chaque élément d'un tableau sont égales à :

- **0** pour des entiers (*int, short, ...*),
- **0.0** pour des nombres à virgule flottante (*double, float*),
- **u0000** pour des caractères (*char*),
- **false** pour des booléens (*boolean*),
- **null** pour des objets (*Object, String*).



Remarques importantes

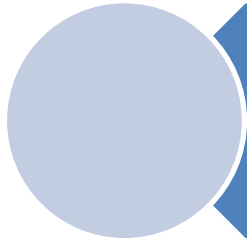
On ne peut pas créer un tableau avec une taille négative.
Une instruction telle que:

Point p [] = new Point [-5];

déclenche une exception **java.lang.NegativeArraySizeException**, laquelle, si elle n'est pas interceptée et traitée provoque l'arrêt brutal du programme (on verra comment traiter les exceptions).

De même, on ne peut accéder à un indice de tableau trop grand (c'est à dire accès en dehors des bornes ou limites du tableau).

Avec: **int tab[] = new int [10];** l'instruction **tab[10] = 12;** déclenche une exception **java.lang.ArrayIndexOutOfBoundsException** .(en fait les indices des éléments d'un tableau varient de 0 à taille - 1).



Création avec un initialiseur

Les tableaux peuvent être initialisés par l'intermédiaire d'une liste de valeurs séparées par une virgule et compris entre des accolades .

```
type [ ] identificateur = { valeur1, ..., valeurN };
```

```
type identificateur [ ] = { valeur1, ..., valeurN };
```

Exemples:

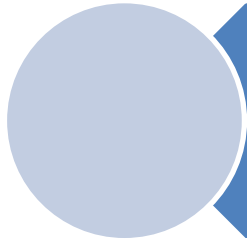
```
int [ ] notes = {10, 9, 12, 14, 16, 15, 17, 20, 19, 18};
```

```
int notes [ ] = {10, 9, 12, 14, 16, 15, 17, 20, 19, 18};
```

L'utilisation d'un initialiseur n'est utilisable que dans une déclaration

```
int [ ] notes;
```

```
notes = {10, 9, 12, 14, 16, 15, 17, 20, 19, 18}; //interdit
```



Utilisation d'un tableau

En Java, on peut utiliser un tableau de deux façons différentes :

- en accédant individuellement à chacun de ses éléments
- en accédant globalement à l'ensemble du tableau .

L'accès individuel aux éléments d'un tableau est réalisé en utilisant ses indices, soit les numéros de chacun de ses éléments, en sachant que le premier commence à l'indice 0.

Exemples:

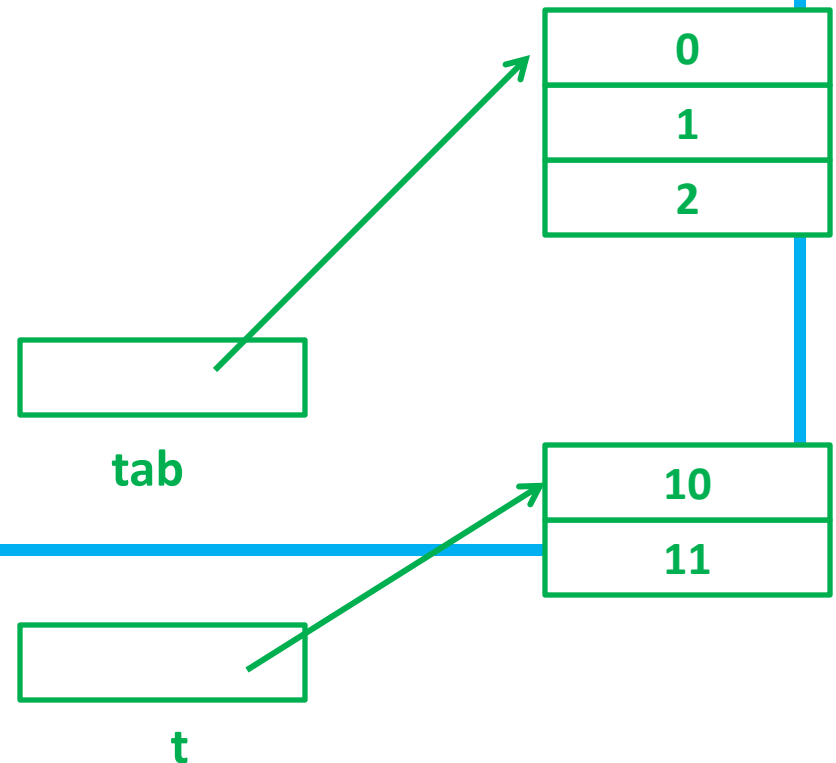
```
String s[ ] = new String [10];  
s[2] = new String ("Bonjour"); //place la chaîne "Bonjour" dans le  
                                // 3eme élément du tableau  
  
double d [ ] = new double [6];  
d[ 5]++;      // incrémente de 1 le dernier élément du tableau
```

Utilisation d'un tableau

La manipulation globale d'un tableau se fait par affectation de tableaux. Il est possible d'affecter une variable de type tableau à une autre variable, à condition qu'elles soient déclarées **avec le même type** de composantes.

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

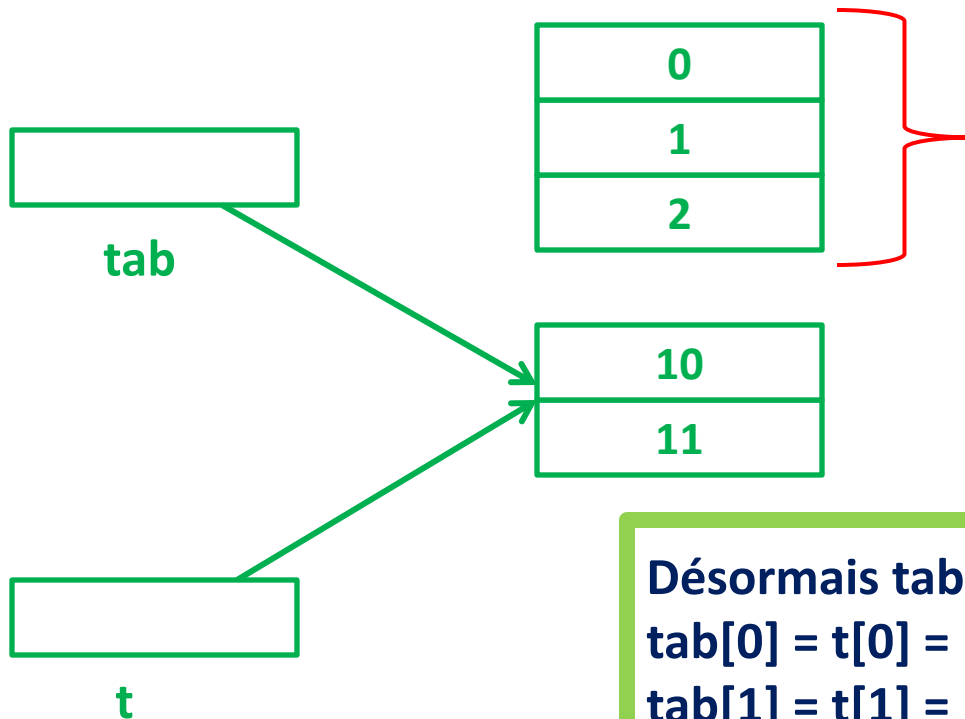
```
int [ ] t = new int [2] ;  
for(int i =0 ; i < 2 ;i++) t[i] = i+10 ;
```



Utilisation d'un tableau

Maintenant avec l'affectation: **tab = t;**

On se retrouve donc dans la situation suivante:



Cet objet sera candidat au Garbage Collector s'il n'est plus référencé.

Désormais **tab** et **t** désignent le même tableau.
tab[0] = t[0] = 10
tab[1] = t[1] = 11

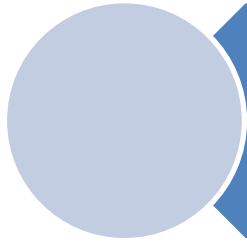
Utilisation d'un tableau

Remarque importante

```
public class Tab01 {  
    public static void main (String[] args) {  
        int t[] = new int [3];  
        t[0]=1; t[1]=2; t[2]=3;  
        int d[] = new int[3];  
        d[0] =1; d[1] =2; d[2] =3;  
        System.out.println (t.equals (d));  
        System.out.println (t == d);  
        t = d; //t et d désignent désormais le même tableau  
        System.out.println (t.equals(d));  
        System.out.println (t == d);  
    }  
}
```

false
false
true
true

Même si deux tableaux contiennent les mêmes éléments (donc même contenu) et sont créés avec deux **new** identiques, il y a deux espaces mémoires différents créés, donc ils ne désignent pas le même tableau..



Utilisation d'un tableau

Taille d'un tableau

On accède à la taille du tableau avec le mot clé **length**.

NB: il ne faut pas confondre cette variable avec la méthode **length()** de la classe **String** qui donne la longueur d'une chaîne de caractères.

```
float notes[ ]= new float [100]; // notes.length vaut 100.
```

Pour parcourir le tableau on peut faire:

```
for( int i = 0; i < notes.length ;i++) //ou bien
```

```
for( int i = 0; i < 100;i++)
```

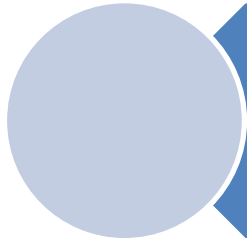
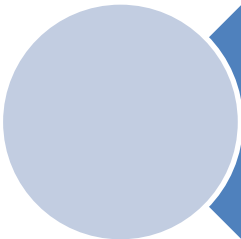



Tableau en argument ou en valeur de retour

Comprenez la transmission d'un tableau en argument ou en valeur de retour d'une méthode comme celle réalisée avec les objets.

Lorsqu'on transmet un nom de tableau en argument d'une méthode, on transmet en fait (une copie de)la référence au tableau .

La méthode agit alors directement sur le tableau concerné et non sur une copie .



Exemple de tableau en argument et en retour

```
public class TabInverse {  
    /*méthode retournant sous forme de tableau  
    l'inverse d'un tableau transmis en argument */  
    public static int[ ] inverseTab (int t[ ])  
    { int tabres[ ] = new int [t.length];  
      for ( int i = t.length -1; i >= 0 ; i--)  
          tabres [ t.length - i - 1] = t[ i ];  
      return tabres;  
    }  
    /*méthode affichant les éléments du tableau renversé*/  
    public static void afficheTab( int t[ ])  
    { for ( int i = 0; i < t.length; i++)  
        System.out.print ( t[ i ]+ " ");  
    }  
}
```

Classe de teste pour l'exemple précédent

```
public class RunTabInverse {  
    public static void main( String args[ ])   
    { int tabAinverser [ ] = new int [5];  
      tabAinverser [0] = 1;  
      tabAinverser [1] = 80;  
      tabAinverser [2] = 71;  
      tabAinverser [3] = 6;  
      tabAinverser [4] = 500;  
      TabInverse ti = new TabInverse();  
      ti.afficheTab(ti.inverseTab (tabAinverser));  
    }  
}
```



500	6	71	80	1
-----	---	----	----	---

Tableau à plusieurs indices

Introduction

Les tableaux vus jusqu'ici sont des **tableaux à une dimension** : conceptuellement tous les éléments se trouvent dans une seule ligne (ou colonne).

Les **tableaux à plusieurs dimensions** sont utiles dans la modélisation des données, mais ce sont les tableaux à deux dimensions qui sont de loin les plus utilisés en informatique. Nous concentrons notre étude à leur cas.

Un **tableau à deux dimensions, ou matrice**, représente un rectangle composé de lignes et de colonnes. Chaque élément stocké dans le tableau est adressé par sa position, donnée par sa ligne et sa colonne.

En Java, si `tab` est un tableau à deux dimensions, l'élément de ligne `i` et colonne `j` est désigné par `tab[i][j]`.

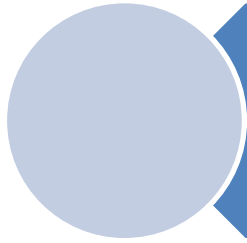


Tableau à deux dimensions

Déclaration

Pour déclarer un tableau à deux dimensions, on peut utiliser l'une de ces trois déclarations qui sont équivalentes :

```
int t [ ] [ ] ; // tableau d'entiers à deux dimensions  
int [ ] t [ ] ; // idem  
int [ ] [ ] t ; // idem
```

Elles déclarent que **t** est une référence à un tableau, dans lequel chaque élément est lui-même une référence à un tableau d'entiers.

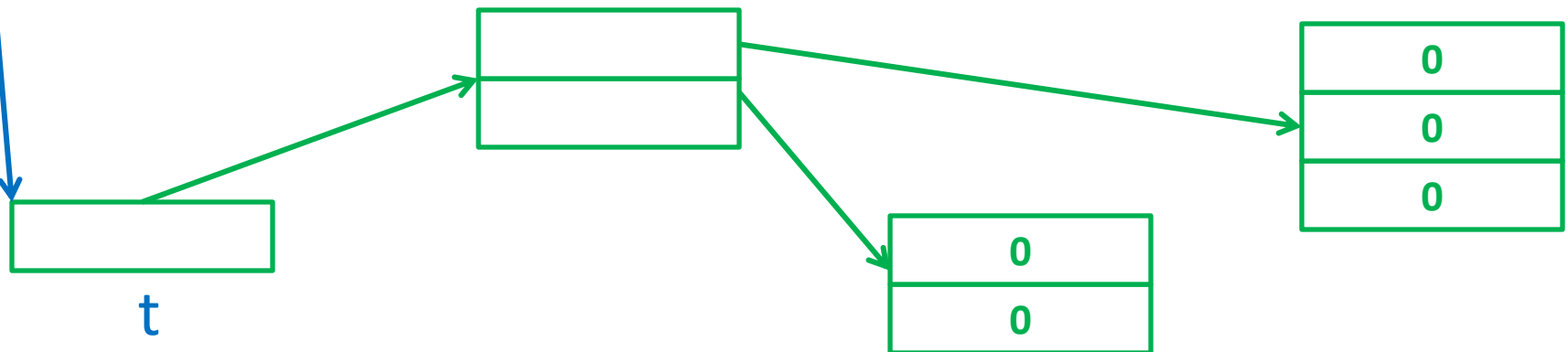
Tableau à deux dimensions

Création

Considérons l'instruction :

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

L'initialiseur de **t** comporte deux éléments dont l'évaluation crée un tableau de 3 entiers et un tableau de 2 entiers . On aboutit à cette situation (les éléments des tableaux sont, comme d'habitude, initialisés à 0) :



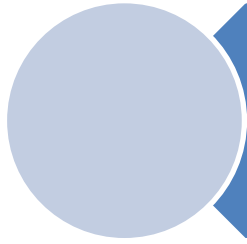


Tableau à deux dimensions

Dans ces conditions, on voit que :

- la notation **$t[0]$** désigne la référence au premier tableau de **3** entiers
- la notation **$t[0][1]$** désigne le deuxième élément de ce tableau
- la notation **$t[1]$** désigne la référence au second tableau de **2** entiers
- la notation **$t[1][i-1]$** désigne le **ième** élément de ce tableau .
- l'expression **$t.length$** vaut **2**
- l'expression **$t[0].length$** vaut **2**
- l'expression **$t[1].length$** vaut **3**

Tableau à deux dimensions

Second exemple :

On peut aboutir à une situation très proche de la précédente en procédant ainsi :

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

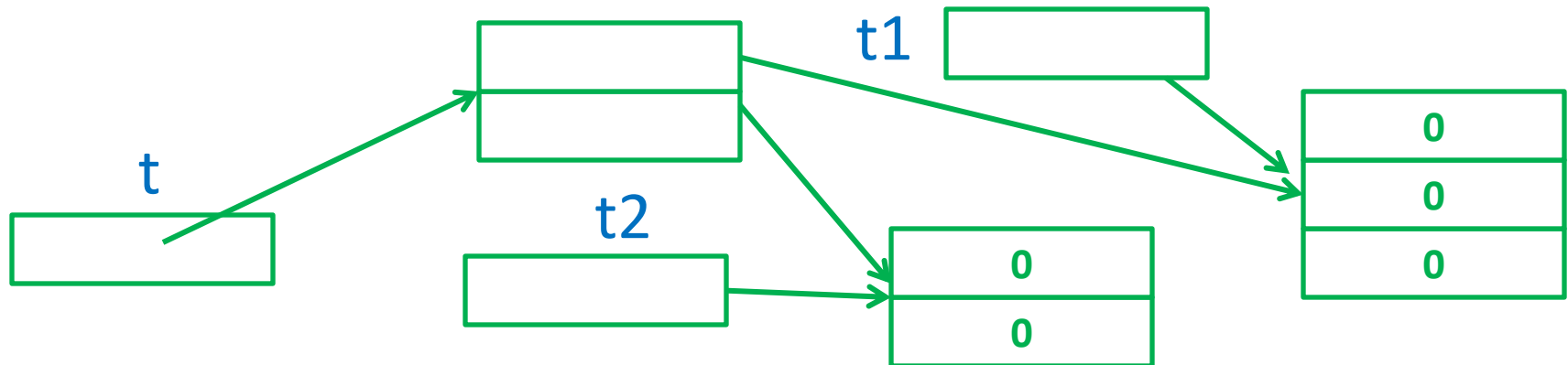
```
t = new int [2][ 3] ;    // création d'un tableau de deux tableaux d'entiers
```

```
int [ ] t1 = new int [3] ; // t1 = référence à un tableau de 3 entiers
```

```
int [ ] t2 = new int [2] ; // t2 = référence à un tableau de 2 entiers
```

```
t[0] = t1 ; t[1] = t2 ;    // on range ces deux références dans t
```

La situation peut être illustrée ainsi :



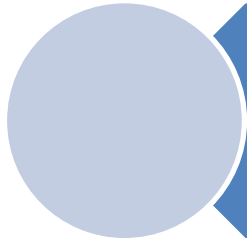
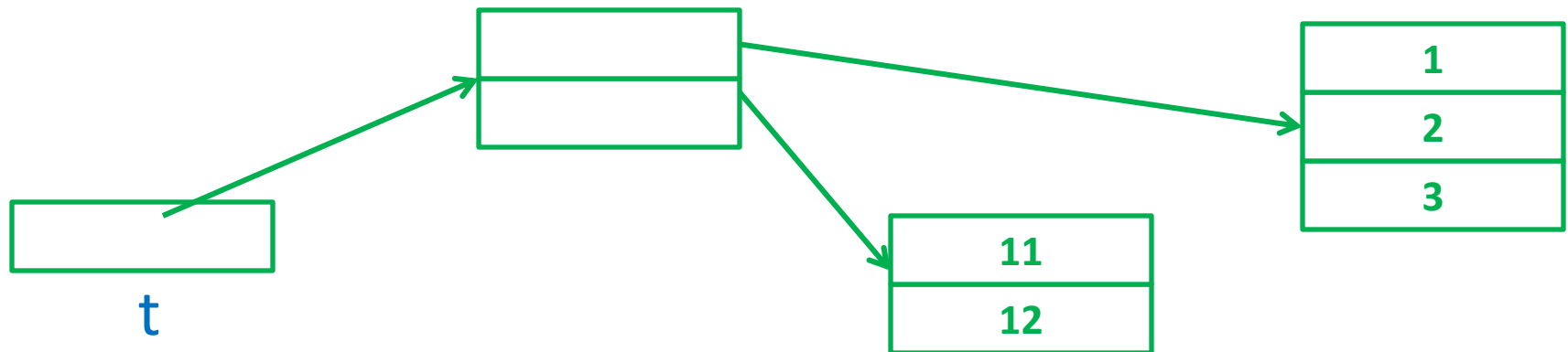


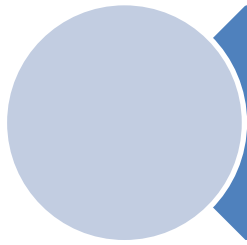
Tableau à deux dimensions

Dans le premier exemple, nous avons utilisé un initialiseur pour les deux références à introduire dans le tableau t; autrement dit, nous procédons comme pour un tableau à un indice . Mais, les initialiseurs peuvent tout à fait s'imbriquer, comme dans cet exemple :

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

ce qui correspond à ce schéma :





Tableaux réguliers

Rien n'empêche que dans un tableau toutes les lignes aient la même taille. Par exemple si l'on souhaite disposer d'une matrice de NLIG lignes et de NCOL colonnes, on peut procéder comme suit:

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

Et faire:

```
for (int i = 0; i<NLIG;i++) tab[i] = new int [NCOL];
```

Mais on peut écrire plus simplement:

```
int tab [ ] [ ] =new int [NLIG][NCOL];
```

Maintenant il sera possible de parcourir ce tableau sans recourir à la variable **length**, comme ceci:

```
for (int i = 0; i<NLIG;i++)  
for (int j = 0; j<NCOL;j++)  
tab[ i ][ j ] = i+j;
```

