# Chapitre 9

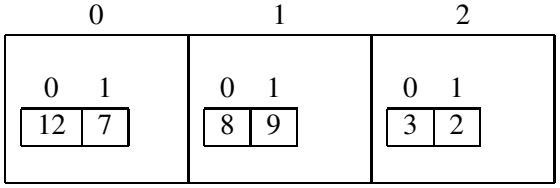
# Tableaux à deux dimensions

## Introduction aux tableaux de tableaux

Nous avons déjà vu qu'on peut utiliser un tableau pour chaque type de données Java de base –`int, double, char, boolean` – ainsi que pour le type des chaînes de caractères `String`.

On peut également utiliser un tableau pour stocker dans chaque case, non pas une valeur simple, mais un tableau d'un certain type. Par exemple, on peut créer un tableau contenant dans chaque case un tableau d'entiers.

On peut représenter un tel tableau de la façon suivante :




Le type de ce tableau s'écrit : `int[][]`. Il n'y a rien là de nouveau ou mystérieux : c'est la règle normale qui s'applique. On prend le type de ce qu'on met dans chaque case du tableau (ici `int[]`) et on ajoute des crochets.

## 9.1 Créer et manipuler des tableaux de tableaux

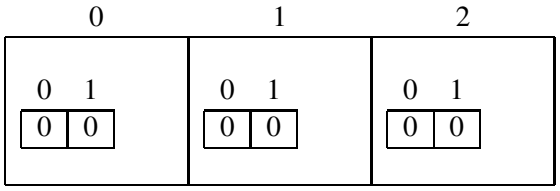
### 9.1.1 Création des tableaux

On peut créer de deux façons des tableaux de tableaux : en créant en même temps le gros tableau et les tableaux contenus dans chaque case ou en créant d'abord le grand tableau et en remplissant ensuite les différentes case.

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

L'instruction `new` crée ici les 4 tableaux : le grand tableau à 3 cases et les 3 petits tableaux à deux cases contenus dans les 3 cases en question. Dans chacune des cases de ces petits tableaux, il y a la valeur par défaut du type `int` : 0.

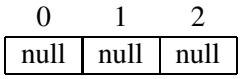
Donc le tableau créé ici est le suivant :




Si l'on souhaite créer seulement le grand tableau, le code à écrire est le suivant :

```
int[][] tab;
tab = new int[3][];
```

Cela crée un tableau à 3 case avec dans chacune, la valeur par défaut du type `int[]`, qui est `null`.




On peut en avoir confirmation en affichant le contenu du tableau.

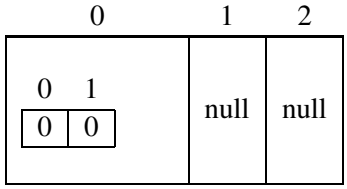
```
public class Tabtab{
   public static void main(String[] args){
      int[][] tab;
      tab = new int[3][];
      for (int i=0; i<tab.length; i=i+1){
         System.out.println(tab[i]);
      }
   }
}
```

```
> java Tabtab
null
null
null
```

On peut ensuite affecter un tableau d'entier à une case. Dans l'instruction suivante, il s'agit d'un nouveau tableau créé par `new`.

```
tab[0] = new int[2];
```

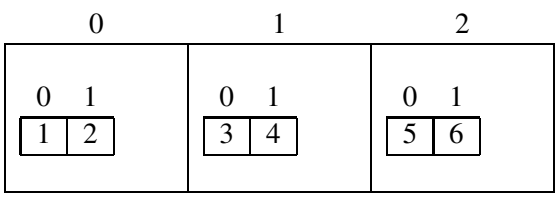
Le tableau devient alors :




Il est possible de définir directement un tableau de tableau lors de la déclaration d'une variable, avec les accolades.

```
int[][] tab = {{1,2},{3,4},{5,6}};
```

Ce qui crée le tableau suivant :



### 9.1.2 Affectations et utilisation des valeurs

On peut utiliser un entier contenu dans une case d'un des petits tableaux ou changer la valeur d'un de ces cases en précisant deux numéros de cases successivement : le numéro du grand tableau puis celui du petit, chacun dans sa paire de crochets.

```
int[][] tab;
tab = new int[3][2];
tab[1][0] = 12;
System.out.println(tab[1][0]);
```

Il est possible également de réaliser une affectation sur l'ensemble d'un petit tableau contenu dans une des cases. Par exemple, on peut affecter le contenu d'une case à une variable de type tableau d'entiers.

```
int[][] tab;
int[] autreTab;
tab = new int[3][2];
autreTab = tab[2];
```

L'affectation dans l'autre sens d'un tableau d'entier à une case de `tab` est également possible.

### 9.1.3 Parcours d'un tableau de tableau

Pour parcourir toutes les cases de tous les petits tableaux, il faut deux boucles imbriquées : une qui parcours les cases du grand tableau, l'autre qui pour chacune des cases du grand tableau, parcourt les cases du petit tableau qu'elle contient.

Le nombre de case du grand tableau est donné par l'attribut `tab.length` et le nombre de cases du petit tableau contenu dans la case numéro `i` est donné par l'attribut `tab[i].length`.

```
int[][] tab;
tab = new int[3][2];
tab[1][0] = 12;
for (int casegrand = 0; casegrand<tab.length; casegrand++){
   for (int casepetit=0; casepetit < tab[casegrand].length;
       casepetit++){
      System.out.print(tab[casegrand][casepetit] + "␣");
   }
   System.out.println();
}
```

## 9.2 Tableau à deux dimensions

Les tableaux de tableaux de java peuvent s'interpréter comme des tableaux à deux dimensions avec des lignes et des colonnes. A ce titre, ils peuvent servir à représenter toutes les informations qu'on a l'habitude de représenter avec des tableaux de ce genre : des planning avec les jours en colonnes et les heures en lignes, des tableaux de ventes avec des produits en colonnes et des zonnes géographiques en lignes, etc.

Prenons comme exemple le tonnage de bateaux coulés et construits pendant la seconde guerre mondiale retranscrit dans le tableau suivant.

| Année | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 |
|---|---|---|---|---|---|---|---|
| Tonnage allié coulé | 810 | 4 407 | 4 398 | 8 245 | 3 611 | 1 422 | 451 |
| Tonnage construit | 332 | 1 219 | 1 964 | 7 182 | 14 585 | 13 349 | 3 834 |

Si l'on ôte les têtes de colonnes et de lignes, les données qui sont des milliers de tonnes de navires, occupent deux lignes et sept colonnes.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 810 | 4 407 | 4 398 | 8 245 | 3 611 | 1 422 | 451 |
| 1 | 332 | 1 219 | 1 964 | 7 182 | 14 585 | 13 349 | 3 834 |

### 9.2.1 Représentation du tableau en Java

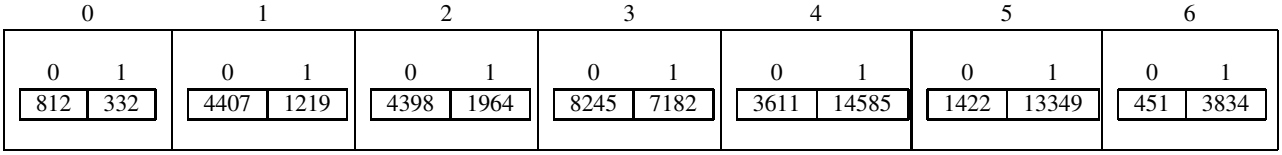
On peut les stocker dans un tableau de tableaux de deux façons : soit un grand tableau à deux cases pour contenir chacune une ligne et dans chaque case un petit tableau à 7 cases pour les 7 années ; soit un grand tableau à 7 cases avec dans chaque case un petit tableau à 2 cases. Selon ce qu'on veut faire avec ces données, l'un ou l'autre choix peut être meilleur. Dans certains cas, le choix est arbitraire.

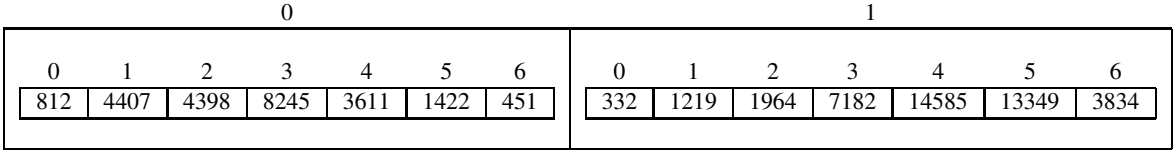
Voyons les deux possibilités écrites en Java.

```
public class Tab2d{
   public static void main(String[] args){
      int[][] tabTonnage_v1 = {{810,332},{4407,1219},{4398,1964},
                               {8245,7182},{3611,14585},{1422,13349},
                               {451,3834}};
      int[][] tabTonnage_v2 = {{810,4407,4398,8245,3611,1422,451},
                               {332,1219,1964,7182,14585,13349,3834}};
      System.out.println(tabTonnage_v1[3][1]);
      System.out.println(tabTonnage_v2[1][3]);
   }
}
```

Le programme affiche le nombre de milliers de tonnes construites (ligne 2) en 1942 (colonne 4). On voit que l'accès à la case se fait avec les deux indices dans deux ordres différents.

Si l'on dessine les tableaux correspondants cela donne ce qui suit.




Dans la suite de la section, nous opterons pour la première version du tableau où les colonnes sont données avant les lignes (cf. `tabTonnage_v1`).

### 9.2.2 Parcours des lignes et des colonnes

Comment calculer le total du tonnage coulé pendant toute la guerre ? Il faut parcourir toutes les cases de la première ligne et faire la somme des entiers qu'elle contient. Une simple boucle permet de parcourir les 7 colonnes.

```
public class Tab2d{
   public static void main(String[] args){
      int[][] tabTonnage_v1 = {{810,332},{4407,1219},{4398,1964},
                               {8245,7182},{3611,14585},{1422,13349},
                               {451,3834}};
      int total = 0;
      for (int colonne=0; colonne<tabTonnage_v1.length; colonne++){
         total = total + tabTonnage_v1[colonne][0];
      }
      System.out.println("Tonnage coulé: " + total);
   }
}
```

En ce qui concerne les colonnes, cela n'aurait pas trop de sens d'ajouter des bateau coulés à des bateaux construits. Ce qui a un sens dans cet exemple, c'est de faire la différence entre bateaux construits et coulés pour avoir ce que l'on peut appeler une variation du tonnage disponible. Par exemple, pour l'année 1942, ce calcul pour s'écrire comme suit :

```
int delta_tonnage_42 = tabTonnage_v1[3][1] -tabTonnage_v1[3][0];
System.out.println("variation du tonnage en 1942: "+
              delta_tonnage_42);
```

On peut calculer cet indice pour chacune des années.

```
for (int col=0; col<tabTonnage_v1.length; col++){
   System.out.println("variation du tonnage en " +
                (1939+col) + ": " +
                (tabTonnage_v1[col][1] -tabTonnage_v1[col][0]));
}
```

Au lieu d'afficher les résultats, on peut les stocker dans un tableau à 7 cases.

```
int[] tab_delta = new int[tabTonnage_v1.length];
for (int col=0; col<tabTonnage_v1.length; col++){
   tab_delta[col] = tabTonnage_v1[col][1] -tabTonnage_v1[col][0];
}
```

### 9.2.3 Conclusion sur les tableaux à deux dimensions

L'usage de tableaux de tableaux pour représenter en Java des tableaux à deux dimensions est très fréquent. C'est même l'usage principal des tableaux de tableaux.

On peut de la même façon représenter des tableaux à trois dimensions par des tableaux de tableaux de tableaux (par exemple, type `int[][][]`), à quatre dimensions, etc. Jusqu'à la dimension 3 on peut avoir une représentation graphique (quoique moins facilement qu'en dimension 2). Au-delà, il

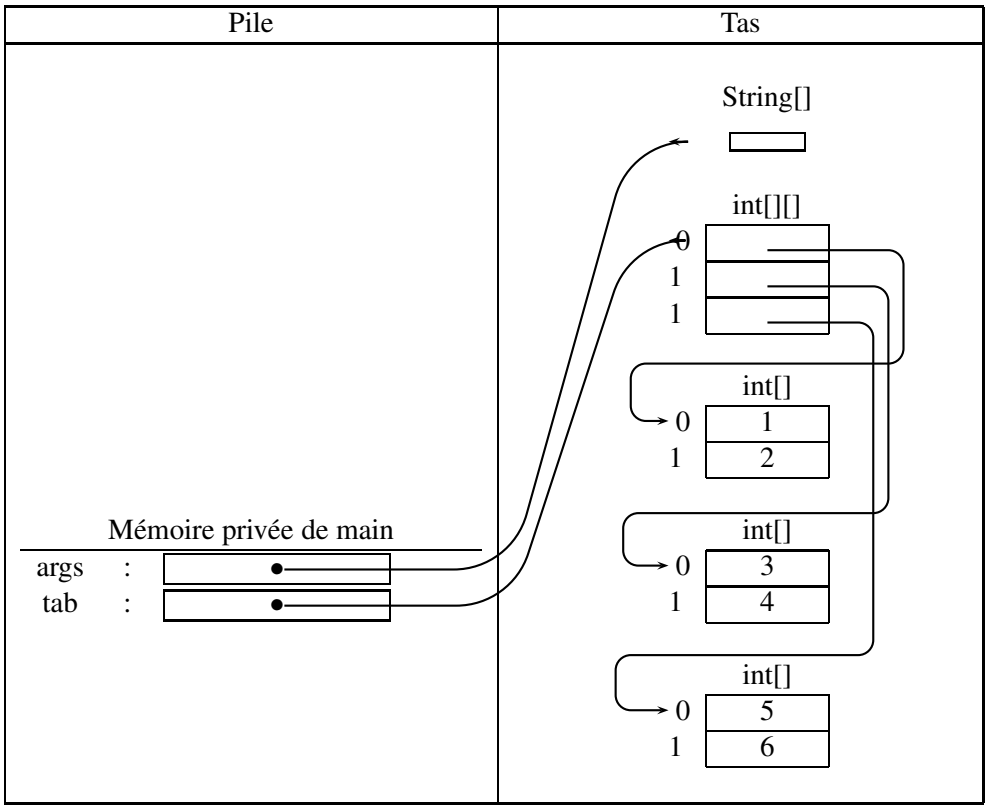
s'agit de structures qui n'ont pas de représentation graphique et sont donc assez abstraites. On parle alors plutôt de vecteurs de dimension 4, 5, et au-delà. On utilise plus rarement ce genre de structures.

## 9.3 Tableaux de tableaux et mémoire

Si l'on déclare un tableau de la façon suivante :

```
int[][] tab = {{1,2},{3,4},{5,6}};
```

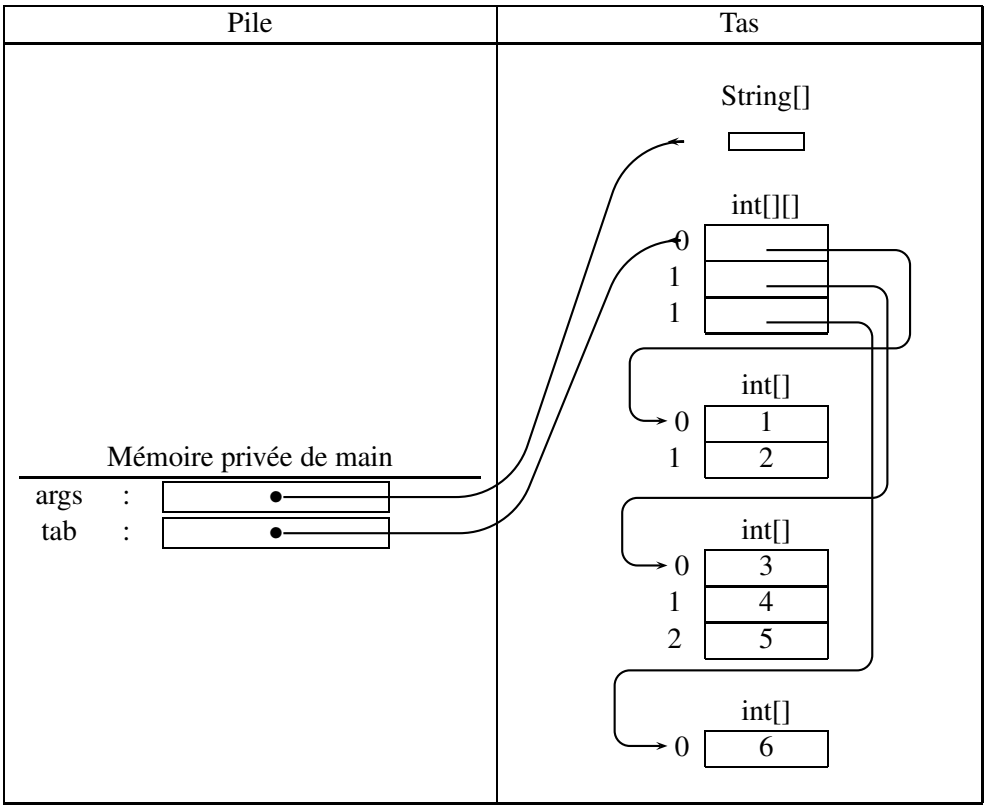
On définit un tableau de trois cases et dans chacune des trois cases il y a un tableau à deux cases. Les cases du grand tableau ne contiennent pas directement les trois petits tableaux mais leurs adresses ou références. De la même façon qu la variable `tab` dans la pile ne contient pas directement le grand tableau, qui est dans le tas, mais son adresse.




### 9.3.1 Tableaux non rectangulaires

Compte tenu de la structure de tableaux enchassés dans un autre tableau, rien n'oblige à ce que les tableaux des différentes cases aient la même taille. On peut parfaitement avoir un tableau avec dans la première case un tableau de taille 2, dans la deuxième case, un tableau de taille 3 et dans la troisième case un tableau de taille 1. Suit la déclaration d'un tel tableau et l'état mémoire correspondant.

```
int[][] tab = {{1,2},{3,4,5},{6}};
```




Est-ce que ces tableaux assymétriques ont un intérêt en pratique ? Oui, parfois.

Par exemple, les distances entre villes sont souvent représentées par un tableau triangulaire : la relation étant symétrique, il est inutile d'avoir une case pour la distance A-B et une autre pour la distance B-A.

L'image suivante a été trouvée sur un site web touristique.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Agadir | | | | | | | | | | | | | |
| Al Hoceima | 1091 | | | | | | | | | | | | |
| Casablanca | 511 | 536 | | | | | | | | | | | |
| Essaouira | 173 | 887 | 351 | | | | | | | | | | |
| Fès | 756 | 275 | 289 | 640 | | | | | | | | | |
| Marrakech | 273 | 758 | 238 | 176 | 483 | | | | | | | | |
| Meknès | 740 | 335 | 229 | 580 | 60 | 467 | | | | | | | |
| Ouarzazate | 375 | 992 | 442 | 380 | 687 | 204 | 652 | | | | | | |
| Oujda | 1100 | 293 | 632 | 983 | 343 | 826 | 403 | 820 | | | | | |
| Rabat | 601 | 445 | 91 | 442 | 198 | 321 | 138 | 528 | 541 | | | | |
| Safi | 295 | 792 | 256 | 129 | 545 | 157 | 486 | 361 | 888 | 347 | | | |
| Tanger | 880 | 323 | 369 | 720 | 303 | 598 | 288 | 811 | 609 | 278 | 625 | | |
| Tétouan | 892 | 278 | 385 | 736 | 281 | 675 | 258 | 820 | 555 | 294 | 641 | 57 | |
| | Agadir | Al Hoceima | Casablanca | Essaouira | Fès | Marrakech | Meknès | Ouarzazate | Oujda | Rabat | Safi | Tanger | Tétouan |

Autre exemple : si l'on veut retracer dans un tableau les notes d'un élève avec une colonne pour chaque matière, rien ne dit que le nombre de contrôles est égal dans toutes les matières. Il peut donc y avoir un nombre de lignes différent d'une colonne à l'autre.

### 9.3.2 Tableaux avec partage

Dans un tableau de tableaux, rien n'empêche de mettre le même petit tableau dans plusieurs case du grand tableau. On a alors une structure étrange, que l'on ne peut pas dessiner sans prendre en compte la notion d'adresse ou référence. Le programme suivant est un exemple de tableau à 3 cases contenant toutes les trois le même petit tableau.

```
public class TabPartage{
   public static void afficher(char[][] t){
      for (int lig=0; lig<t[0].length; lig=lig+1){
         for (int col=0; col<t.length; col=col+1){
            System.out.print(t[col][lig]+" ");
         }
         System.out.println();
      }
      System.out.println();
   }
   public static void main(String[] args){
      char[] petit = {'a','b','c','d'};
      char[][] grand = new char[3][];
      grand[0]=petit;
      grand[1]=petit;
      grand[2]=petit;
      afficher(grand);
      grand[0][1]='X';
      afficher(grand);
   }
}
```

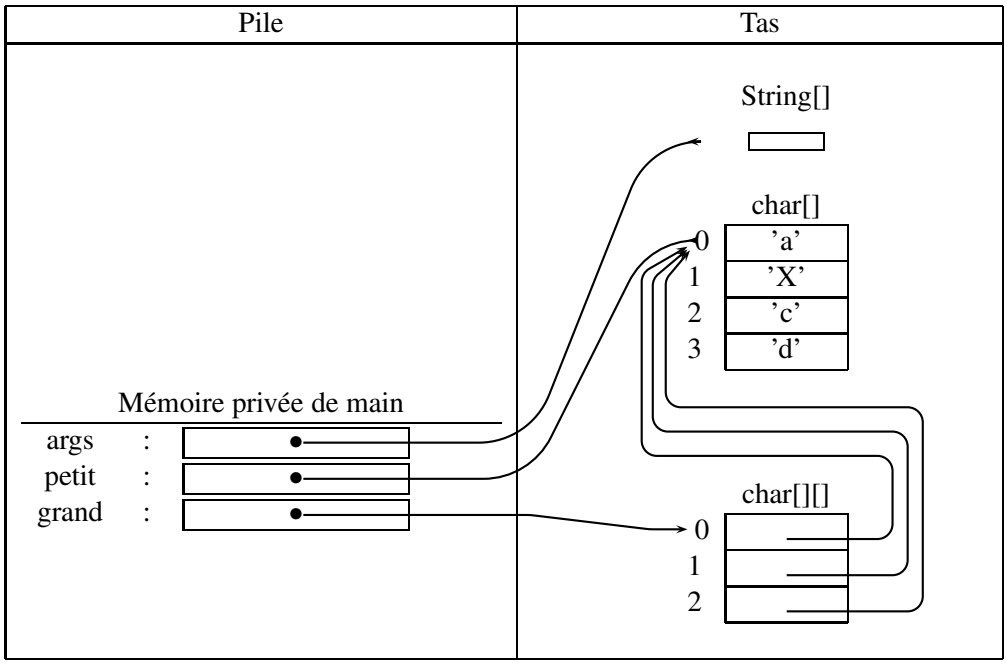
L'exécution du programme produit l'affichage suivant :

```
> java TabPartage
a a a
b b b
c c c
d d d

a a a
X X X
c c c
d d d
```

On voit ici que l'affectation du caractère `'X'` à `grand[0][1]` a changé aussi `grand[1][1]` et `grand[2][1]` qui sont trois noms différents qui désignent le même emplacement dans le tas.

L'état de la mémoire après l'affectation est dessiné ci-dessous.




## 9.4 Conclusion

Il n'y a rien de très nouveau dans ce chapitre. La seule réelle nouveauté est la possibilité de créer d'un coup tous les tableaux d'une structure à plusieurs dimensions avec un new suivi de plusieurs paires de crochets comme dans `new int[3][5]`.

Pour le reste, la façon d'accéder aux cases, les affectations autorisées, la façon d'obtenir la taille d'un tableau se font de la même façon que d'habitude.

Les tableaux à deux dimensions servent très souvent à représenter des données dans des applications de gestion. Ils permettent d'implémenter les matrices utilisées dans des applications de simulation numérique.

Las tableaux à deux dimensions sont également très utiles pour représenter les jeux de damier (échecs, dames, reversi), les jeux sur feuille (morpion, bataille navale, sudoku, mots croisés).