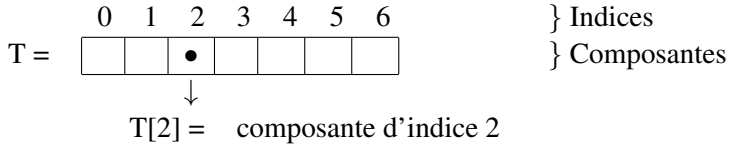
Chapitre 4

Tableaux

Jusqu'ici, nous avons employé les variables pour stocker les valeurs *individuelles* de types primitifs : une variable de type `int` pour stocker un entier, une variable de type `boolean` pour un booléan, etc.

Un **tableau** est *une structure regroupant plusieurs valeurs de même type*, appelées *composantes* du tableau. On peut traiter un tableau comme un tout, ou composante par composante. Traité comme un tout, on pourra le stocker dans une variable, le passer en paramètre ou le donner en résultat d'un calcul. Chaque *composante* est désignée individuellement via son *indice*, qui correspond à sa position dans le tableau, et peut être traitée comme *variable individuelle* : on pourra consulter sa valeur, la modifier, etc.



T[0], T[1], T[2], T[3], ..., T[6] } 7 Variables (composantes)

Les tableaux sont des structures des données présentes dans tous les langages de programmation. En général, chaque langage possède un *type tableau prédéfini* avec une syntaxe spécifique.

4.1 Déclaration et création

En Java, avant d'utiliser un tableau, il faut :

1. **Déclarer** une variable de type tableau (symbole `[]`), en indiquant le type T de ses futures composantes ;

```
T [] tab;          // tab est declare tableau de T
```

2. **Créer explicitement** la structure du tableau en mémoire (opération `new`), en donnant sa taille et le type T de ses éléments. Cette taille ne pourra plus changer : en Java les tableaux sont de *taille fixe*.

```
tab = new T[taille];
```

3. L'initialisation des composantes avec des valeurs par défaut, est réalisée implicitement par l'opération de création.

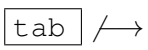
Étudions plus en détail ces étapes.

Déclaration

L'instruction :

```
T [] tab;
```

déclare une variable `tab` *destinée à contenir* un tableau, dont les composantes seront de type T. Après déclaration, la variable `tab` existe, mais **n'est pas encore initialisée** à un tableau. En Java, on dira que `tab` **ne référence pas encore** [1] de tableau. Le dessin suivant montre l'état de `tab` en mémoire :



où le symbole $\not\longrightarrow$ est lu : "n'est pas initialisé".

Exemples :

```
int []  tabNum;      // tabNum est un tableau avec composantes de type int
double [] t;         // t est un tableau avec composantes de type double
String [] m;         // m est un tableau avec composantes de type String
tabNum[0] = 5;       // provoque une erreur: le tableau n'existe pas
```

Après ces déclarations, les variables `tabNum`, `t` et `m` existent, mais pas encore la suite de composantes que chacune d'entre elles pourra désigner. Par exemple, il est impossible de modifier la première composante de `tabNum` (notée `tabNum[0]`) : elle n'existe pas encore. Le compilateur signale l'erreur :

```
Test.java:7: variable tabNum might not have been initialized
    tabNum[0] = 5;
    ^
```

Création

L'opération de création :

```
new T[n];
```

réalise la création et l'initialisation d'un tableau de n composantes de type T :

1. *Allocation en mémoire* d'un espace suffisant pour stocker n composantes de type T.
2. *Initialisation* des composantes du tableau avec des valeurs par défaut.

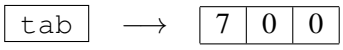
Les tableaux en Java sont de taille fixe. Une fois le tableau créé, l'espace qui lui est alloué en mémoire ne peut pas changer. Par exemple, il est impossible d'ajouter ou d'enlever des composantes d'un tableau.

Exemple 1 : Déclaration, puis création du tableau `tab` avec trois entiers.

1. La notion de référence sera abordée dans le chapitre dédié aux objets.

```
int []  tab;          // Declaration
tab = new int[3];     // Creation
tab]0] = 7;
```

Après l'instruction de création `new`, la variable `tab` est initialisée (ou, *fait référence*) à un tableau contenant trois entiers. Après l'affectation `tab[0] = 7`, la structure du tableau en mémoire est :



Il est possible de réunir la déclaration et la création d'un tableau en une seule instruction. On pourra ainsi déclarer et créer le tableau de l'exemple 1 par :

Exemple 2 : Déclaration et création en une seule instruction.

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

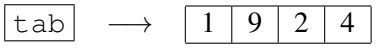
Initialisation par une liste de valeurs

Lorsqu'un tableau est de petite taille, il est possible de l'initialiser en donnant la liste des valeurs de chaque composante. On utilise la notation $\{v_0, v_1, \ldots v_n\}$, où v_i est la valeur à donner à la composante i du tableau. Nous reprendrons souvent cette notation pour **regrouper en une seule instruction** la déclaration, la création et l'initialisation d'un tableau.

Exemple 3 : Déclaration, création et initialisation d'un tableau en une seule instruction.

```
int []  tab = {1,9,2,4};
```

Il est alors **inutile de réaliser une création explicite** via `new` : elle se fait automatiquement à la taille nécessaire pour stocker le nombre des valeurs données. En mémoire on aura :



Valeurs par défaut à la création

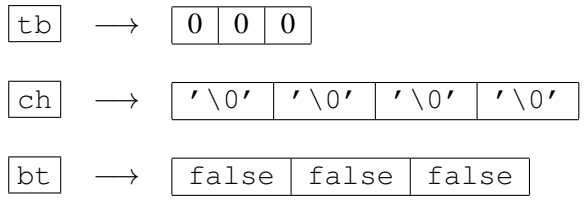
Lors de la création, les composantes d'un tableau sont initialisées avec des valeurs par défaut :
- les composantes `boolean` sont initialisées à `false`.
- les composantes numériques sont initialisées à 0.
- les composantes `char` sont initialisées au caractère nul `'\0'`.
- les composantes *référence* [2] sont initialisées à la valeur `null` (référence nulle).

Exemple 4 : Valeurs par défaut dans les composantes après création.

```
int []  tb = new int[3];
char [] ch = new char[4];
boolean [] bt = new boolean[3];
```

2. Voir le chapitre dédié aux objets.

L'opération `new` initialise les composantes de ces tableaux par :

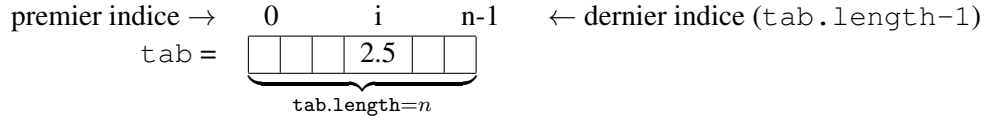


Longueur d'un tableau

La *taille* ou *longueur* d'un tableau est le nombre n des composantes qu'il contient. Supposons que `tab` désigne un tableau de taille n. On peut obtenir sa longueur par la notation `tab.lenght`. Les indices du tableau `tab` sont alors compris entre 0 et `tab.length-1`. Cette notation sera souvent employée pour fixer la valeur maximale qui peut prendre l'indice d'un tableau (voir exemple 6).

4.2 Accès aux composantes

L'accès à une composante de tableau permet de traiter cette composante comme n'importe quelle variable individuelle : on peut modifier sa valeur dans le tableau, l'utiliser pour un calcul, un affichage, etc. L'accès d'une composante se fait via son *indice ou position* dans le tableau. En Java, la première position a pour indice 0, et la dernière, a l'indice n-1 (taille du tableau moins un).



tab[i] vaut 2.5

L'accès aux composantes de `tab` n'a de sens que pour les indices dans l'intervalle [0,. . ., tab.length-1]. Si i est un indice compris dans cet intervalle :

- `tab[i]` : est un **accès à la composante de position i dans** `tab`. On peut consulter ou modifier cette valeur dans le tableau. *Exemple* : `tab[i] = tab[i] + 1;`
- **accès en dehors des bornes du tableau** `tab` : si `j` n'est pas compris entre 0 et tab.length-1, l'accès `tab[j]` provoque une erreur à l'exécution : l'indice j et la composante associée n'existent pas dans le tableau. Java *lève l'exception* `ArrayIndexOutOfBoundsException`.

Exemple 5 : Modification d'une composante, accès en dehors des bornes d'un tableau.

```
public class Test {
    public static void main (String args[]) {
        double [] tab = {1.0, 2.5, 7.2, 0.6}; // Creation
        // Affichage avant modification
        Terminal.ecrireString("tab[0] avant = ");
        Terminal.ecrireDoubleln(tab[0]);
        tab[0] = tab[0] + 4;
        // Affichage apr\'es modification
```

```
        Terminal.ecrireString("tab[0]␣apres␣=␣");
        Terminal.ecrireDoubleln(tab[0]);
        // tab[5] = 17; // Erreur: indice en dehors des bornes
    }
}
```

Ce programme affiche la valeur de la première composante de `tab`, avant et après modification.

```
Java/Essais> java Test
tab[0] avant = 1.0
tab[0] apres = 5.0
```

Si nous enlevons le commentaire de la ligne 11, l'exécution se termine par une erreur : l'indice 5 est en dehors des bornes du tableau.

```
Java/Essais> java Test
tab[0] avant = 1.0
tab[0] apres = 5.0
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Test.main(Test.java:9)
```

Exemple 6 : Parcours pour affichage d'un tableau `tab`. La boucle fait varier l'indice de `tab` entre `i= 0` et `i <= tab.lenght-1`.

```
public class AfficheTab {
    public static void main (String args[]) {
        int[] tab = {10,20,30,40};
        for (int i=0; i<= tab.length -1; i++) {
            Terminal.ecrireStringln("tab["+ i + "]␣=␣␣" + tab[i]);
        }
    }
}
```

Ce programma affiche :

```
Java/Essais> java AfficheTab
tab[0] =  10
tab[1] =  20
tab[2] =  30
tab[3] =  40
```

Une erreur commune dans une boucle, est de fixer le dernier indice à `i <= tab.length` plutôt qu'à `i <= tab.length -1`. Si nous changeons la condition de la boucle de cette manière, l'exécution produit une erreur : l'indice `tab.length`, égal à 4 ici, n'existe pas dans `tab`.

```
Java/Essais> java AfficheTabErr
tab[0] =  10
tab[1] =  20
tab[2] =  30
tab[3] =  40
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
        at AfficheTabErr.main(AfficheTabErr.java:5)
```

Exemple 7 : Boucles d'initialisation et d'affichage. On souhaite initialiser un tableau avec des notes d'élèves saisies au clavier. Le programme demande le nombre de notes à saisir, et créé un tableau `lesNotes` de cette taille. Une première boucle initialise le tableau ; la boucle suivante affiche son contenu. Les itérations se font dans l'intervalle de `i=0` jusqu'à `i <= lesNotes.length-1`.

```
public class Notes {
    public static void main (String args[]) {
        int nombreNotes;
        Terminal.ecrireString("Nombre␣de␣notes␣a␣saisir?␣");
        nombreNotes = Terminal.lireInt();
        double [] lesNotes = new double[nombreNotes];
        // Initialisation
        for (int i=0; i<= lesNotes.length -1; i++) {
            Terminal.ecrireString("Note␣no.␣" + (i+1) + "?␣");
            lesNotes[i] = Terminal.lireDouble();
        }
        // Affichage
        Terminal.sautDeLigne();
        Terminal.ecrireStringln("Notes␣dans␣le␣tableau:");
        Terminal.ecrireStringln("**********************");
        for (int i=0; i<= lesNotes.length -1; i++) {
            Terminal.ecrireString("Note␣no.␣" + (i+1) + "␣=␣");
            Terminal.ecrireDoubleln(lesNotes[i]);
        }
    }
}
```

```
Java/Essais> java Notes
Nombre de notes a saisir? 4
Note no. 1? 7.6
Note no. 2? 11
Note no. 3? 14
Note no. 4? 5

Notes dans le tableau:
**********************
Note no. 1 = 7.6
Note no. 2 = 11.0
Note no. 3 = 14.0
Note no. 4 = 5.0
```

Affectation entre tableaux

Il est possible d'affecter une variable de type tableau à une autre autre variable, à condition qu'elles soient déclarées avec le même type de composantes. Après affectation, *les deux variables référent à un même et seul tableau en mémoire* : elles deviennent *synonymes*. Toute modification sur un composant de l'une modifie le même composant de l'autre.

Exemple 8 :

```
        int [] t;
```

```
int [] m = {2,3,4,5,6};
t = m; // t et m designent un meme tableau
Terminal.ecrireStringln("t[0] = " + t[0]);
Terminal.ecrireStringln("m[0] = " + m[0]);
// Modification de t[0]
t[0] = 9;
// Nouveaux t[0], m[0]
Terminal.ecrireStringln("Nouveau t[0] = " + t[0]);
Terminal.ecrireStringln("Nouveau m[0] = " + m[0]);
```

L'affectation de `m` dans `t` a pour *effet de bord* de rendre ces deux variables *synonymes* : elles *référent* toutes les deux au même espace mémoire, qui contient le tableau $\{2, 3, 4, 5, 6\}$ désigné par `m`. L'affectation `t[0] = 9` est ainsi "visible" lors d'un accès à `m`. Ce programme affiche :

```
t[0] = 2
m[0] = 2
Nouveau t[0] = 9
Nouveau m[0] = 9
```

Enfin, les tableaux désignés par les variables d'une affectation peuvent avoir des longueurs différentes.

Exemple 9 : Affectation entre tableaux de tailles différentes.

```
int [] t = {10, 20};
int [] m = {2,3,4,5,6};
Terminal.ecrireStringln("Longueur de t = " + t.length);
t = m; // t contient maintenant un tableau de 5 elements
// Nouvelle longueur de t
Terminal.ecrireStringln("Nouvelle longueur de t = " + t.length);
```

Ce programme affiche :

```
Longueur de t = 2
Nouvelle longueur de t = 5
```

Égalité entre tableaux

Lorsqu'on compare deux tableaux par des tests *d'égalité* (==) ou *d'inégalité* (!=), le test porte sur *l'identité des tableaux et non sur leur contenu*. Cela signifie qu'on cherche à savoir, non pas si les tableaux contiennent les mêmes éléments, mais s'ils ont été crées par un seul et même `new`. Voyons un exemple.

```
public class Tab5{
    public static void main(String[] argv){
        int[] tab1;
        int[] tab2;
        int[] tab3;
        tab1 = new int[3];
        tab1[0] = 10;
        tab1[1] = 20;
        tab1[2] = 30;
        tab2 = new int[3];
        tab2[0] = 10;
```

```
        tab2[1] = 20;
        tab2[2] = 30;
        tab3 = tab2;
        if (tab1 == tab2){
            Terminal.ecrireStringln("tab1 et tab2 sont egaux");
        }else{
            Terminal.ecrireStringln("tab1 et tab2 sont differents");
        }
        if (tab2 == tab3){
            Terminal.ecrireStringln("tab2 et tab3 sont egaux");
        }else{
            Terminal.ecrireStringln("tab2 et tab3 sont differents");
        }
    }
}
```

L'exécution du programme donne :

```
> java Tab5
tab1 et tab2 sont différents
tab2 et tab3 sont égaux
```

Explication : `tab1` et `tab2` sont deux tableaux dont les contenus sont égaux, mais ces deux tableaux ont été créés par deux `new` différents. Il s'agit donc de deux espaces mémoires distincts. Changer l'un ne change pas l'autre. En revanche, `tab2` et `tab3` sont créés par un seul et unique `new`. Changer le contenu de l'un change aussi le contenu de l'autre. Il s'agit d'un seul et unique espace mémoire.

4.3 Exemples avec tableaux unidimensionnels

Exemple 10 : Recherche du minimum et maximum dans un tableau d'entiers. Deux variables `min` et `max` sont initialisées avec le premier élément du tableau. La boucle de recherche des minimum et maximum, compare chaque élément avec ces deux valeurs. La comparaison se fait à partir du deuxième élément : c'est pourquoi l'indice i débute à `i=1`.

```
public class minMax {
    public static void main (String args[]) {
        int n;
        Terminal.ecrireString("Combien des nombres a' saisir? ");
        n = Terminal.lireInt();
        int [] tab = new int[n];
        // Initialisation
        for (int i=0; i<= tab.length -1; i++) {
            Terminal.ecrireString("Un entier? ");
            tab[i] = Terminal.lireInt();
        }
        // Recherche de min et max
        int min = tab[0];
        int max = tab[0];
        for (int i=1; i<= tab.length -1; i++) {
            if (tab[i] < min) { min = tab[i];}
            if (tab[i] > max) { max = tab[i];}
```

```
        }
        Terminal.ecrireStringln("Le␣minimum␣est:␣" + min);
        Terminal.ecrireStringln("Le␣maximum␣est:␣" + max);
    }
}
```

Voici une exécution du programme :

```
Java/Essais> java minMax
Combien des nombres a` saisir? 5
Un entier? 5
Un entier? 2
Un entier? -6
Un entier? 45
Un entier? 3
Le minimum est: -6
Le maximum est: 45
```

Exemple 11 : Gestion de notes. Nous modifions le programme de l'exemple 7 afin de calculer la moyenne des notes, la note minimale et maximale, et le nombre de notes supérieures ou égales à 10. Nous reprenons la boucle de recherche du minimum et maximum de l'exemple 10.

```
public class Notes {
    public static void main (String args[]) {
        int nombreNotes;
        Terminal.ecrireString("Nombre␣de␣notes␣a␣saisir?␣");
        nombreNotes = Terminal.lireInt();
        double [] lesNotes = new double[nombreNotes];
        // Initialisation
        for (int i=0; i<= lesNotes.length -1; i++) {
              Terminal.ecrireString("Note␣no.␣" + (i+1) + "?␣");
              lesNotes[i] = Terminal.lireDouble();
        }
        double min = lesNotes[0];
        double max = lesNotes[0];
        double somme = 0;
        int sup10 = 0;
        for (int i=0; i<= lesNotes.length -1; i++) {
            if (lesNotes[i] < min) { min = lesNotes[i];}
            if (lesNotes[i] > max) { max = lesNotes[i];}
            if (lesNotes[i] >= 10) { sup10++;}
            somme = somme + lesNotes[i];
        }
        Terminal.ecrireString("La␣moyenne␣des␣notes␣est:␣");
        Terminal.ecrireDoubleln(somme/nombreNotes);
        Terminal.ecrireStringln("Le␣nombre␣de␣notes␣>=␣10␣est:␣" + sup10);
        Terminal.ecrireStringln("La␣note␣minimum␣est:␣" + min);
        Terminal.ecrireStringln("La␣note␣␣maximum␣est:␣" + max);
    }}
```

```
Java/Essais> java Notes
Nombre de notes a` saisir? 4
```

```
Note no. 1? 5
Note no. 2? 8
Note no. 3? 10
Note no. 4? 15
La moyenne des notes est: 9.5
Le nombre de notes >= 10 est: 2
La note minimum est: 5.0
La note  maximum est: 15.0
```

Exemple 12 : Inversion d'un tableau de caractères. La boucle d'inversion utilise deux variables d'itération i et j, initialisées avec le premier et le dernier élément du tableau. A chaque tour de boucle, les éléments dans i et j sont échangés, puis i est incrémenté et j décrémenté. Il y a deux cas d'arrêt possibles selon la taille du tableau : s'il est de taille impair, alors l'arrêt se produit lorsque i=j ; s'il est de taille pair, alors l'arrêt se fait lorsque `j < i`. En conclusion, la boucle doit terminer si `i >= j`.

```
public class Inversion {
    public static void main (String[] args) {
        int n;
        char [] t;
        Terminal.ecrireString("Combien␣de␣caracteres␣a␣saisir?␣");
        n = Terminal.lireInt();
        t = new char[n];
        // Initialisation
        for (int i=0; i<= t.length-1; i++) {
                Terminal.ecrireString("Un␣caractere:␣␣");
                t[i] = Terminal.lireChar();
        }
        // Affichage avant inversion
        Terminal.ecrireString("Le␣tableau␣saisi:␣");
        for (int i=0; i<= t.length-1; i++) {
                Terminal.ecrireChar(t[i]);
        }
        Terminal.sautDeLigne();
        // Inversion: arret si (i >= j)
        int i,j;
        char tampon;
        for (i=0, j= t.length-1 ; i < j; i++, j--) {
                tampon = t[i];
                t[i] = t[j];
                t[j] = tampon;
        }
        // Affichage final
        Terminal.ecrireString("Le␣tableau␣inverse:␣");
        for (int k=0; k<= t.length-1; k++) {
                Terminal.ecrireChar(t[k]);
        }
        Terminal.sautDeLigne();
    }
}
```

Un exemple d'exécution :

```
Java/Essais> java Inversion
Combien de caracteres a saisir? 5
Un caractere:  s
Un caractere:  a
Un caractere:  l
Un caractere:  u
Un caractere:  t
Le tableau saisit: salut
Le tableau inverse: tulas
```

4.4 Le tri par sélection

Les programmes de tri de tableaux sont couramment utilisés dans la gestion des données. Nous présentons un algorithme simple de tri, le *tri par sélection*. Il est appelé ainsi car, lors de chaque itération, il sélectionne l'élément le plus petit parmi ceux *restant à trier* et le met à sa place dans le tableau, en *l'échangeant avec l'élément* qui s'y trouve. Nous illustrons cette méthode sur le tableau $\{4, 5, 1, 9, 8\}$.

La première fois, l'algorithme sélectionne le plus petit du tableau, qui est 1, et l'échange avec l'élément qui se trouve à la première place (4) :

$$\begin{array}{rl} & \{\underline{4}, 5, \boxed{1}, 9, 8\} \\ \text{après échange } \Rightarrow & \{\boxed{1}, 5, \underline{4}, 9, 8\} \end{array}$$

Au bout de cette première itération, le premier élément est trié : c'est le plus petit de tout le tableau. Les éléments restant à trier sont ceux à partir de la 2ème position. La fois suivante, le plus petit parmi eux (4), est sélectionné et échangé avec l'élément se trouvant à la deuxième place (5) :

$$\begin{array}{rl} & \{1, \underline{5}, \boxed{4}, 9, 8\} \\ \text{après échange } \Rightarrow & \{1, \boxed{4}, \underline{5}, 9, 8\} \end{array}$$

Après la deuxième itération, les deux premiers éléments sont triés : la première place contient le plus petit, la deuxième, contient le deuxième plus petit. L'algorithme finit lorsqu'il ne reste plus qu'un seul seul élément à trier, qui se trouve nécessairement à sa place : la dernière. Terminons l'application de l'algorithme :

$$\begin{array}{rll} & \{1, 4, \boxed{5}, 9, 8\} : & \text{5 est échangé avec lui-même} \\ \Rightarrow & \{1, 4, 5, \underline{9}, \boxed{8}\} : & \text{8 est échangé avec 9} \\ \Rightarrow & \{1, 4, 5, 8, 9\} : & \text{Le tableau est trié} \end{array}$$

Algorithme de tri par sélection :

Entrée : un tableau tab taille n
Sortie : le tableau tab trié.

Pour chaque indice i de `tab` compris dans [0, . . ., n-2], faire :

1. Sélectionner le plus petit élément parmi ceux d'indices [i, . . ., n-1], et déterminer son indice `Im`.
2. Échanger les éléments `tab[Im]` et `tab[i]`.

□

Le pas de sélection du plus petit élément se fait également avec une boucle. La technique employée est similaire à celle de l'exemple 10, mais ici, la recherche ne se fait pas sur tout le tableau, mais seulement sur la partie restant à trier lors de chaque itération.

Exemple 13 : Tri par sélection.

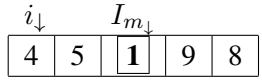
```
public class triSelection {
    public static void main (String args[]) {
        int n;
        int [] tab;
        Terminal.ecrireString("Nombre␣de␣entiers␣a␣trier?␣");
        n = Terminal.lireInt();
        tab = new int[n];
        // Initialisation de tab
        for (int i = 0; i <= tab.length -1; i++) {
            Terminal.ecrireString("Un␣entier?␣");
            tab[i] = Terminal.lireInt();
        }
        // Tri
        for (int i = 0; i <= tab.length-2; i++) {
            // Recherche du min dans [i .. tab.lentgh-1]
            int Im = i;
            for (int j = i+1 ; j <= tab.length-1; j++) {
                if (tab[j] < tab[Im]) {Im = j;}
            }
            // Echange de tab[i] avec le min trouve
            int tampon = tab[i];
            tab[i] = tab[Im];
            tab[Im] = tampon;
        }
        // Affichages
        Terminal.ecrireString("Tableau␣trie:␣");
        for (int i = 0; i <= tab.length -1; i++) {
            Terminal.ecrireString("␣␣");
            Terminal.ecrireInt(tab[i]);
        }
        Terminal.sautDeLigne();
    }
}
```

Voici un exemple d'exécution :

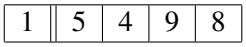
```
Java/Essais> java triSelection
Nombre d'entiers a trier? 6
Un entier? 10
Un entier? 2
Un entier? 7
Un entier? 21
Un entier? 8
Un entier? 5
Tableau trie:  2  5  7  8  10  21
```

Nous présentons une trace partielle de ce programme, sur le tableau `tab = {4,5,1,9,8 }`. Elle montre, pour chaque pas de boucle, la position de l'indice i, et celle de l'indice I_m du plus petit élément parmi ceux restant à trier :

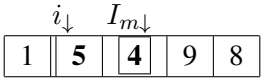
Itération 1 : `i = 0`, `Im` = 2.



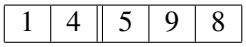
Après échange :



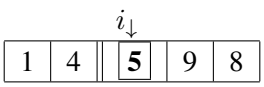
Itération 2 : `i = 1`, `Im` = 2.



Après échange :

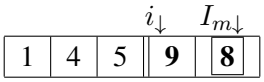


Itération 3 : `i = 2`, `Im` = 2.

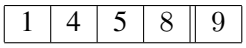


Après échange : rien ne change.

Itération 4 : `i = 3`, `Im` = 4.



Après échange, il ne reste plus qu'un élément : le tableau est donc trié.

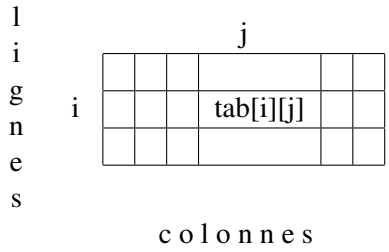


4.5 Tableaux à deux dimensions

Les tableaux vus jusqu'ici sont des *tableaux à une dimension* : conceptuellement tous les éléments se trouvent dans une seule ligne (ou colonne, cela revient au même). 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 stockée 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]`.



= `tab`, où `tab[i][j]` ⇒ valeur à la position (i,j)

Déclaration

En Java, un tableau de n dimensions et composantes de type T est déclaré par :

```
T [] [] $\ldots$ []  tab;     // n fois le symbole []
```

Chaque ajout du symbole [] permet d'obtenir une dimension supplémentaire :

```
int [] t;           // une dimension
int [] [] m;        // deux dimensions
char [] [] [] p;    // trois dimensions
```

Création

Comme avant, nous utilisons l'opération `new` de création de tableaux, en donnant en plus du type des éléments, la taille de chaque dimension.

Exemple 14 : Création d'un tableau d'entiers à deux dimensions, avec 3 lignes et 5 colonnes. Après création, nous modifions l'élément de la ligne 1 et colonne 2 par `t[1][2] = 7`.

```
int [][] t = new int [3][5];      //  3 lignes, 5 colonnes
t[1][2] = 7;
```

Nous pouvons imaginer `t` comme un rectangle avec 3 lignes et 5 colonnes. Par convention, la première dimension est celle des lignes, et la deuxième, celle des colonnes. Comme avant, les indices débutent à 0. Lors de sa création, les éléments du tableau sont initialisés à 0 par défaut.

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	7	0	0
2	0	0	0	0	0

Longueur d'une dimension

Il est également possible d'obtenir la longueur d'une des dimensions du tableau : nombre des lignes pour la première dimension, nombre de colonnes pour la deuxième. La notation :

- `t.length` : donne la longueur de la première dimension, c.a.d., le nombre de lignes du tableau.
- `t[i].length` : donne la longueur de la ligne i de t, autrement dit, le nombre de colonnes de cette ligne.

Exemple 15 :

```
int [][] t = new int [3][5];          //  3 lignes, 5 colonnes
Terminal.ecrireIntln(t.length);       // affiche 3
Terminal.ecrireIntln(t[1].length);    // affiche 5
```

Initialisation

En Java, les tableaux à plusieurs dimensions sont définis à l'aide de tableaux des tableaux. Ce sont des tableaux dont les composantes sont elles-mêmes des tableaux. Comme avant, il est possible d'initialiser une matrice en donnant la liste de ses composantes. Par exemple, un tableau de 3 lignes et 4 colonnes sera initialisé par une suite de trois tableaux, un pour chaque ligne. Chacun des trois tableaux sera composé de 4 éléments : c'est le nombre de colonnes dans chacune des lignes.

Exemple 15 : Initialisation d'une matrice (de 3 lignes et 4 colonnes), par énumération de ses composantes.

```
int [][] tab = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
int [] t = tab[1];
for (int j = 0; j<= t.length-1; j++) {
    Terminal.ecrireInt(t[j]);
}
Terminal.sautDeLigne();
```

Cet exemple nous permet de comprendre qu'un tableau à deux dimensions de tailles n et m, est en réalité un tableau unidimensionnel de n lignes, où chaque ligne est un tableau de m composantes. Le programme affiche :

```
Java/Essais> java Test
5678
```

Parcours des matrices

Les parcours des matrices se font souvent avec des boucles imbriquées : une boucle externe pour parcourir les lignes, pour des indices compris entre 0 et mat.lenght-1, et une boucle interne qui, pour chaque ligne, fait le parcours des éléments dans toutes les colonnes de cette ligne. Les indices des colonnes seront alors compris entre 0 et mat[i].lenght-1.

Exemple 16 : Initialisation d'une matrice par saisie de sa taille et de ses éléments.

```
public class initMatrice {
    public static void main (String args[]) {
```

```
        int n,m;
        Terminal.ecrireString("Nombre␣de␣lignes?␣");
        n = Terminal.lireInt();
        Terminal.ecrireString("Nombre␣de␣colonnes?␣");
        m = Terminal.lireInt();
        int [][] mat = new int [n][m];
        // Initialisation
        for (int i=0; i<= mat.length -1; i++) {
            for (int j=0; j<= mat[i].length -1; j++) {
                Terminal.ecrireString("Element␣(" + i + ",␣" + j + ")?␣");
                mat[i][j] = Terminal.lireInt();
            }
        }
    }
}
```

Ce programme affiche :

```
Java/Essais> java initMatrice
Nombre de lignes? 2
Nombre de colonnes? 3
Element (0, 0)? 1
Element (0, 1)? 2
Element (0, 2)? 3
Element (1, 0)? 4
Element (1, 1)? 5
Element (1, 2)? 6
```

Exemple 17 : Gestion des notes par élèves. Ce programme initialise une matrice de n élèves avec m notes par élève, puis calcule dans un tableau de taille n, la moyenne de chaque élève. Les notes de l'élève i se trouvent à la ligne i de la matrice `notes`, alors que sa moyenne est dans `moyennes[i]`.

```
public class matriceNotes {
    public static void main (String args[]) {
        int n,m;
        Terminal.ecrireString("Nombre␣d'eleves?␣");
        n = Terminal.lireInt();
        Terminal.ecrireString("Nombre␣de␣notes␣par␣eleve?␣");
        m = Terminal.lireInt();
        double [][] notes = new double [n][m];
        double [] moyennes = new double [n];
        // Initialisation
        for (int i=0; i<= notes.length -1; i++) {
            Terminal.ecrireStringln("Notes␣pour␣l'eleve␣"+ (i+1)+ "?");
            for (int j=0; j<= notes[i].length -1; j++) {
                Terminal.ecrireString("␣␣␣␣Note␣"+ (j+1) + "?␣");
                notes[i][j] = Terminal.lireDouble();
            }
        }
       // Calcul des moyennes
        for (int i=0; i<= notes.length -1; i++) {
            for (int j=0; j<= notes[i].length -1; j++) {
```

```
                moyennes[i] = moyennes[i] + notes[i][j];
            }
            moyennes[i] = moyennes[i]/notes[i].length;
        }
        //Affichages
        for (int i=0; i<= moyennes.length -1; i++) {
            Terminal.ecrireString("Moyenne␣de␣l'eleve␣"+ (i+1) + "=␣");
            Terminal.ecrireDoubleln(moyennes[i]);
        }
    }
}
```

Ce programme affiche :

```
Java/Essais> java matriceNotes
Nombre d'eleves? 3
Nombre de notes par eleve? 2
Notes pour l'eleve 1?
   Note 1? 2
   Note 2? 2
Notes pour l'eleve 2?
   Note 1? 6
   Note 2? 17
Notes pour l'eleve 3?
   Note 1? 10
   Note 2? 15
Moyenne de l'eleve 1= 2.0
Moyenne de l'eleve 2= 11.5
Moyenne de l'eleve 3= 12.5
```