

# Exemple introductif

Une table de multiplication

On souhaite afficher par programme les premières lignes d'une table de multiplication.

0	0	0	0	0	...	0	0
0	1	2	3	4	...	9	10
0	2	4	6	8	...	18	20
0	3	6	9	12	...	27	30
:	:	:	:	...	:	:	:
0	9	18	27	36	...	81	90

Séquence de code Java correspondante :

```
Deug.println("Table de multiplication");
for (int i = 0; i < 10; i++) {
    for (int j = 0; j <= 10; j++)
        Deug.print(i*j + " ");
    Deug.println();
}
```

En Java, plutôt que d'afficher des résultats de cette forme, on peut les garder en mémoire en utilisant un

un « tableau de 10 tableaux de 11 entiers ».

Chacun des 10 tableaux représente en fait une ligne de la table de multiplication, et une ligne est elle-même un tableau de 11 entiers.

D'un point de vue mathématique, cela correspond à une matrice : un tableau rectangulaire à deux dimensions.

La séquence de code Java suivante

```
int[ ][ ] mult = new int [10][11];
```

1. déclare une variable mult de type « *tableau de tableaux d'entiers* »
2. **alloue en mémoire** (au moyen de l'opérateur new) l'espace nécessaire pour stocker un tableau de 10 tableaux de 11 entiers (c'est-à-dire une matrice rectangulaire d'entiers de 10 lignes et 11 colonnes)
3. initialise la variable mult : après cette affectation sa valeur fait référence au tableau.

Rappel : **allouer en mémoire** signifie rechercher et réserver l'espace nécessaire en mémoire pour stocker « quelque chose ». La taille de l'espace alloué dépend de la nature du « quelque chose ».

On peut alors dire que :

1. l'expression mult[i] fait référence à un tableau de 11 entiers.

Elle représente la  $i + 1$ -ième ligne de la table de multiplication (en effet la numérotation commence à 0).

Si t est une variable de type *tableau d'entiers* déclarée sous la forme

```
int[ ] t;
```

l'instruction

```
t = mult[i];
```

est correcte et lui affecte une valeur qui fait référence au tableau d'entiers mult[i]

2. mult[i][j] s'évalue comme t[j] avec l'affectation précédente de t

L'emplacement mult[i][j] peut contenir une valeur entière. On va y placer la valeur " $i$  multiplié par  $j$ "

On peut réaliser les opérations habituelles sur les éléments du tableau.

Sur notre exemple, pour donner leurs valeurs aux différents éléments du tableau auquel mult fait référence, et afficher la table complète, on pourra écrire :

```
.....
int[ ][ ] mult = new int [10][11];
.....
for (int i = 0; i < 10; i++)
    for (int j = 0; j <= 10; j++)
        mult[i][j] = i * j;
Deug.println("Table de multiplication");
for (int i = 0; i < 10; i++) {
    for (int j = 0; j <= 10; j++)
        Deug.print(mult[i][j] + " ");
    Deug.println();
}
```

## Deuxième exemple

Relevé de notes d'une classe

Il apparaît sur le papier comme un tableau de la forme suivante :

Matière	Maths	Physique	Info	Anglais
Pierre	12	09	14	13
Marie	15	14	14	10
Paul	06	10	08	16
Jacques	10	11	12	10
Nathalie	08	11	07	09

Chaque colonne correspond à une matière et chaque ligne à un étudiant.

Une ligne donne la liste des notes dans chaque matière pour un étudiant : cette liste correspond en fait à un tableau à une dimension de valeurs numériques :

Notes de Paul	[0]	[1]	[2]	[3]
	06	10	08	16

On représente le relevé de notes sous forme d'un tableau de tableaux, qu'on appellera notes.

Le tableau des notes de Paul (qui est le troisième étudiant dans le tableau papier) correspond donc à notes[2] dans le tableau informatique utilisé.

	<i>(étudiant)</i>	<i>(tableau de ses notes)</i>			
<u>notes[0]</u>	<i>(Pierre)</i>	[0]	[1]	[2]	[3]
		12	09	14	13
<u>notes[1]</u>	<i>(Marie)</i>	[0]	[1]	[2]	[3]
		15	14	14	10
<u>notes[2]</u>	<i>(Paul)</i>	[0]	[1]	[2]	[3]
		06	10	08	16
<u>notes[3]</u>	<i>(Jacques)</i>	[0]	[1]	[2]	[3]
		10	11	12	10
<u>notes[4]</u>	<i>(Nathalie)</i>	[0]	[1]	[2]	[3]
		08	11	07	09

La note d'anglais de Paul (16) correspond alors à la valeur de notes[2][3].

On peut réaliser les opérations habituelles sur cette variable :

- affectation : notes[2][3]=15;
- utilisation : somme = somme + notes[2][3];

# Syntaxe générale de déclaration d'un tableau

```
T[ ]...[ ] t;
```

déclare une variable `t` de type tableau de tableaux de ... de tableaux d'éléments de type `T`

```
t = new Type[dim_1]...[dim_n];
```

alloue un tableau de `dim_1` tableaux de... de `dim_n` tableaux d'éléments de type `T` et affecte à `t` la référence à l'espace correspondant

On dit souvent que ce tableau est de dimension `n`

Pour la  $k$ -ième dimension, les éléments du tableau sont numérotés de 0 à `dim_k-1`.

On accède par `length` aux différentes dimensions. Ainsi,

```
int[ ][ ] notes = new int[5][4];
```

- déclare une variable `notes` de type tableau de tableaux d'entiers,
- alloue un tableau de 5 tableaux de 4 entiers et
- affecte sa référence à la variable `notes`.

Après cette affectation,

`notes.length` vaut 5 :

c'est un tableau de 5 "tableaux de 4 entiers"

`notes[0].length` vaut 4 (c'est un tableau de 4 entiers)

## Exemple

Voici un morceau de programme qui stocke le tableau papier suivant

	Maths	Physique
Pierre	12	09
Paul	06	10
Jacques	10	11

sous la forme d'un tableau à deux dimensions avec les conventions suivantes :

- la première colonne correspond aux *maths* et la seconde à la *physique*
- les lignes correspondent successivement à *Pierre*, *Paul* et *Jacques*.

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

```
notes[0][0] = 12;
```

```
notes[0][1] = 09;
```

```
notes[1][0] = 06;
```

```
notes[1][1] = 10;
```

```
notes[2][0] = 10;
```

```
notes[2][1] = 11;
```

# Saisie d'un tableau à deux dimensions

Sans fonction intermédiaire :

```
public static void Saisie(int[ ][ ] t) {
    int nbEleves = t.length;
    int nbExam = t[0].length;
    for (int i = 0; i<nbEleves; i++) {
        Deug.println("notes de l'eleve" + i);
        for (int j = 0; j<nbExam; j++) {
            Deug.println("note a l'exam" + j);
            t[i][j] = Deug.readInt();
        }
    }
}
```

Avec fonction intermédiaire :

```
public static void SaisieEleve(int[ ] t1) {
    int nbExam = t1.length;
    for (int j = 0; j<nbExam; j++) {
        Deug.println("note a l'exam" + j);
        t1[j] = Deug.readInt();
    }
}
```

```
public static void Saisie(int[ ][ ] t) {
    int nbEleves = t.length;
    for (int i = 0; i<nbEleves; i++) {
        Deug.println("notes de l'eleve" + i);
        SaisieEleve(t[i]);
    }
}
```

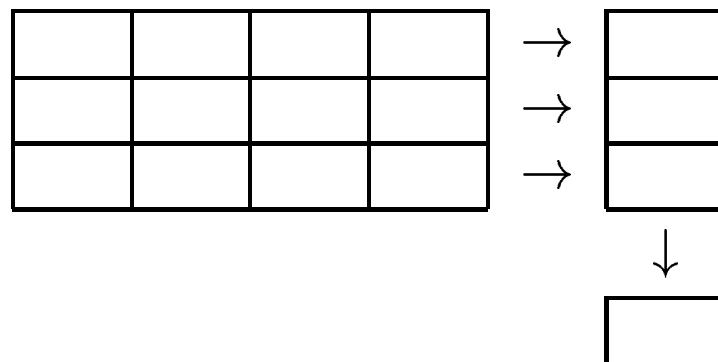
Exemple d'utilisation :

```
public static void main(String[ ] arg) {
    int nbEleves, nbExam;
    Deug.println("Nombre d'élèves");
    nbEleves = Deug.readInt();
    Deug.println("Nombre de notes");
    nbExam = Deug.readInt();
    int[ ][ ] tab = new int[nbEleves][nbExam];
    Saisie(tab);
    ...
}
```

# Exercices

1. Écrivez un programme qui réalise l'affichage d'un tableau à deux dimensions représentant un relevé de notes.
2. Écrivez une fonction qui calcule la moyenne des notes d'*un* étudiant, données sous la forme d'un tableau à une dimension.

En déduire une fonction qui calcule l'ensemble des moyennes pour tous les étudiants sous forme d'un tableau, puis une fonction qui calcule la moyenne générale de tous les étudiants.



# Approfondissement : allocation partielle

On a décrit comment allouer un tableau de `dim_1...` éléments de type `Type`.

Il est également possible d'allouer un tableau de `dim_1...` éléments eux-mêmes de type tableau :

```
Type[ ][ ][ ] t;  
t = new Type[dim_1][ ][ ];
```

alloue un tableau de `dim_1` tableaux de tableaux d'éléments de type `Type`.

Un tableau notes rectangulaire ayant 7 lignes et 4 colonnes peut donc être alloué de la manière suivante :

```
int[ ][ ] notes = new int[7][ ];  
for (int i = 0; i < 7; i++)  
    notes[i] = new int[4];
```

Cela permet de dissocier les allocations des différentes lignes, et donc de définir des tableaux de forme non nécessairement rectangulaire.

# Exemple

```
public class TableauTriangulaire {
    public static void main(String[ ] args) {
        int ligne, col;
        int[ ][ ] tab ;
        tab = new int[10][ ];

        for (ligne = 0; ligne<10; ligne++) {
            tab[ligne] = new int[ligne+1];
            for (col = 0; col<tab[ligne].length; col++)
                tab[ligne][col] = col;
        }

        for (ligne = 0; ligne<10; ligne++)
            Deug.print(tab[ligne].length + " ");
        Deug.println();
        Deug.println("-----");
        for (ligne = 0; ligne<10; ligne++) {
            for (col = 0; col<tab[ligne].length; col++)
                Deug.print(tab[ligne][col] + " ");
            Deug.println();
        }
    }
}
```

## Exemple (suite)

Le programme précédent d'afficher le triangle :

```
1 2 3 4 5 6 7 8 9 10
```

```
-----
```

```
0
```

```
0 1
```

```
0 1 2
```

```
0 1 2 3
```

```
0 1 2 3 4
```

```
0 1 2 3 4 5
```

```
0 1 2 3 4 5 6
```

```
0 1 2 3 4 5 6 7
```

```
0 1 2 3 4 5 6 7 8
```

```
0 1 2 3 4 5 6 7 8 9
```

### Exercice

Écrire un programme permettant de créer et d'afficher un tableau représentant les  $n$  premières lignes du triangle de Pascal.

# Création d'un tableau à l'intérieur d'une fonction

On peut créer un tableau dans une autre fonction que `main`.

Cela peut être utile pour :

- effectuer des calculs intermédiaires
- créer un nouveau tableau qui sera retourné par la fonction.

## Calculs intermédiaires

On considère une suite d'entiers définie par récurrence par :

$$x_{n+10} = a_0x_n + a_1x_{n+1} + \dots + a_9x_{n+9},$$

où  $a_0, \dots, a_9$  sont fixés, ainsi que  $x_0, \dots, x_9$ .

Voici un programme qui affiche la valeur de  $x_n$ , pour  $n \geq 0$ , sans perdre les données initiales :

```

class Suite {
    static int suite (int[ ] a, int[ ] x, int n) {
        int[ ] tmp = new int[x.length]; int futur = 0;
        if (n<0) Deug.exit();
        if (n<10) return x[n];
        for (int i = 0; i<x.length; i++)
            tmp[i] = x[i];
        for (int i = x.length; i<=n; i++) {
            futur = 0; // calcul de x_i
            for (int j = 0; j<x.length; j++)
                futur += a[j]*tmp[j];
            for (int k = 0; k<x.length-1; k++) // mise a jour
                tmp[k] = tmp[k+1]; // de tmp
            tmp[x.length-1] = futur;
        }
        return futur;
    }
}

public static void main(String[ ] args) {
    int[ ] x = new int[10], a = new int[10]; int n;
    for (int i = 0; i<10; i++) {
        Deug.print("a_" + i + " = "); // saisie
        a[i] = Deug.readInt(); // valeurs
        Deug.print("x_" + i + " = "); // initiales
        x[i] = Deug.readInt(); // et coeffs
    }
    Deug.print("indice a calculer : ");
    n = Deug.readInt();
    Deug.println("x_" + n + " = " + suite(a, x, n));
}
}

```

## Valeur renvoyée

On peut aussi se servir d'un tableau comme valeur de retour d'une fonction.

Par exemple pour calculer la somme de 2 matrices de même taille :

```
static double[ ][ ] somme(double[ ][ ] M,
                           double[ ][ ] N) {
    double[ ][ ] P = new double[M.length][M[0].length];

    for (int i = 0; i<M.length; i++)
        for (int j = 0; j<M[0].length; j++)
            P[i][j] = M[i][j]+N[i][j];
    return P;
}
```

# Matrices et images

Le but de la série d'exercices qui suit est de faire des opérations élémentaires sur des images.

## Noir & Blanc

On considère des images en noir et blanc que l'on identifie à des matrices (=tableaux à 2 dimensions) de booléens.

Une case `false` de la matrice correspond à un pixel blanc, une case `true` à un pixel noir.

Exemple : La fonction `init` renvoie une image blanche dont les dimensions ont été fournies en paramètres.

```
static boolean[ ][ ]init(int longueur,int largeur) {
    boolean[ ][ ] im;
    im = new boolean[longueur][largeur];
    return im;
}
```

Exercice : Modifier la fonction `init` pour que toute image ait un cadre noir.

Le passage de la matrice à l'image peut se faire par :

```
static void dessine(boolean[ ][ ] im) {
    for (int i = 0; i<im.length; i++)
        for (int j = 0; j<im[i].length; j++)
            if(im[i][j])
                Deug.drawPoint(i,j);
}
```

# Exercices

Écrire les fonctions suivantes :

1. une fonction qui prend en argument deux entiers  $m$  et  $n$  et renvoie une image  $m \times n$  comprenant une croix.
2. une fonction qui “développe” un négatif (c’est-à-dire inverse noir et blanc sur l’image passée en paramètre).
3. une fonction qui renvoie la superposition de 2 images supposées de même taille.
4. une fonction qui renvoie le miroir d’une image par rapport à l’axe vertical central.
5. [TP] une fonction qui “rogne” une image : elle crée une image qui est la partie utile de l’image d’origine (la partie utile de l’image est assimilée à la couleur noir, le blanc étant le fond de l’image).

## Niveaux de gris

La classe `Deug` fournit une fonction qui permet de préciser le niveau de gris souhaité :

```
public static void setGray(int c);
```

Le paramètre `c` représente l'intensité de la couleur (0 pour noir et 255 pour blanc).

On considère maintenant des images en niveaux de gris. Une image est donc représentée par une matrice d'entiers courts (`short`). La valeur d'une case de la matrice représente l'intensité du pixel correspondant.

# Exercices

1. Écrire une fonction `dessine` ayant pour paramètre une matrice de `short` et qui dessine l'image correspondante.
2. [TP] On veut mettre en place un filtre d'anti-aliasing. Pour cela, on va créer une nouvelle image. L'intensité de la couleur d'un pixel de celle-ci sera obtenu en combinant l'intensité du pixel correspondant dans l'ancienne image et la moyenne des intensité des pixels alentour, dans une proportion  $\frac{5}{6}$  /  $\frac{1}{6}$ . Écrire la fonction correspondante.