

# ***IFT 1015 - Tableaux***

---

Professeur:  
Stefan Monnier

B. Kégl, S. Roy, F. Duranleau, S. Monnier  
Département d'informatique et de recherche opérationnelle  
Université de Montréal

hiver 2006

[Tasso:9] et [Niño: 22.1]

- Concept des tableaux
- Déclaration et instanciation
- Indexation
- Tableaux en Java
- Bornes des tableaux
- Limite des tableaux
- Tableaux d'objets
- Tableaux à plusieurs dimensions

# Stockage d'information

- *Limite* des types simples ou des objets tels que présentés jusqu'ici: **on ne peut traiter une quantité arbitraire d'information à la fois.**
- Et si la quantité d'éléments à manipuler et à conserver en mémoire n'était pas connue au moment de la compilation?

**Ex.:** Trier des nombres lus au clavier.

- Soit un programme qui manipule 8 colis postal à la fois. On aurait les déclarations suivantes:

```
ColisPostal c1, c2, c3, c4, c5, c6, c7, c8;
```

Et si on voulait maintenant 9 colis? 1000?

# Stockage d'information

- *Limite* des types simples ou des objets tels que présentés jusqu'ici: **on ne peut traiter une quantité arbitraire d'information à la fois.**
- Et si la quantité d'éléments à manipuler et à conserver en mémoire n'était pas connue au moment de la compilation?

**Ex.:** Trier des nombres lus au clavier.

- Soit un programme qui manipule 8 colis postal à la fois. On aurait les déclarations suivantes:

```
ColisPostal c1, c2, c3, c4, c5, c6, c7, c8;
```

Et si on voulait maintenant 9 colis? 1000?

→ On a besoin d'une façon de stocker des ensembles arbitraires d'information ⇒ **les tableaux.**

## Tableau

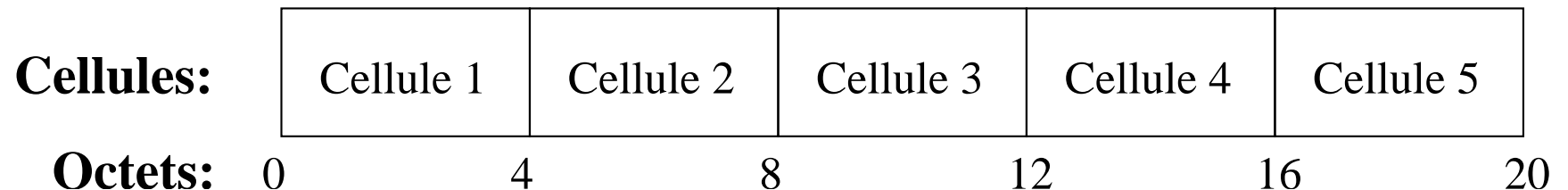
Une structure composée d'une séquence contiguë de variables, toutes de *même type*.

## Cellule

Élément d'un tableau. Il s'agit du sous-ensemble de l'espace mémoire du tableau où est stocké l'une de ses variables.

## Exemple:

Un tableau de 5 entiers.



En Java, on crée un tableau ainsi:

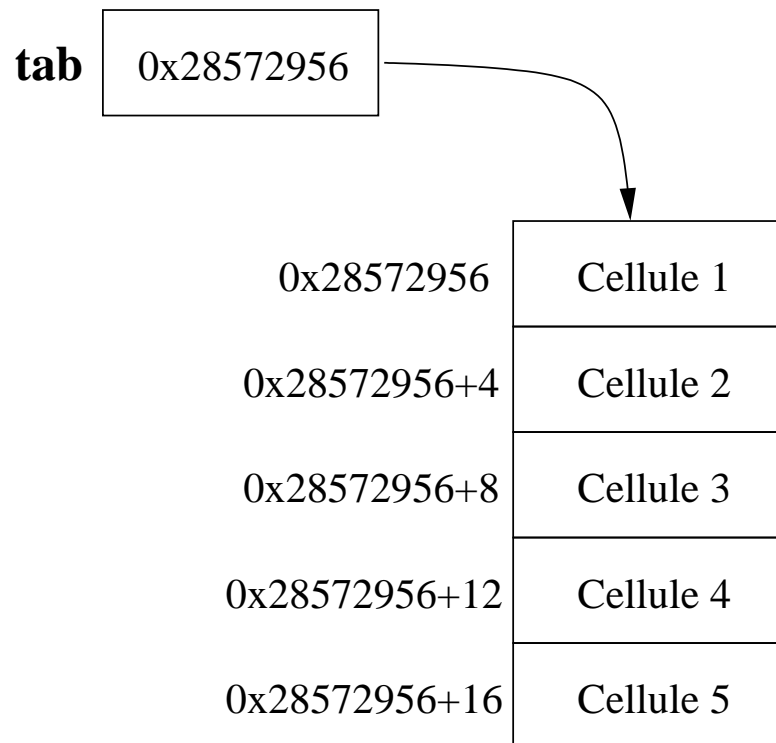
```
Type[] nomDuTableau = new Type[combien];
```

où

- `Type` est le type de chaque cellule du tableau.
- `combien` est une *expression* de type *entier* qui indique combien de cellules le tableau contiendra. Cette valeur doit être  $\geq 1$ .
- `nomDuTableau` est une *référence* sur l'espace mémoire totale du tableau.

# Exemple

```
int[] tab = new int[5];
```



La première cellule est au début du bloc de mémoire; la seconde est 4 octets plus loin; la troisième 8, ...

# Initialisation

En Java, la valeur initiale de chaque cellule est 0 pour les nombres, le caractère 0 pour les caractères, ou `null` pour les objets (y compris les chaînes de caractères).

**Avertissement:** ce n'est pas le cas de tous les langages!!

Il est possible d'instancier un tableau avec des valeurs initiales:

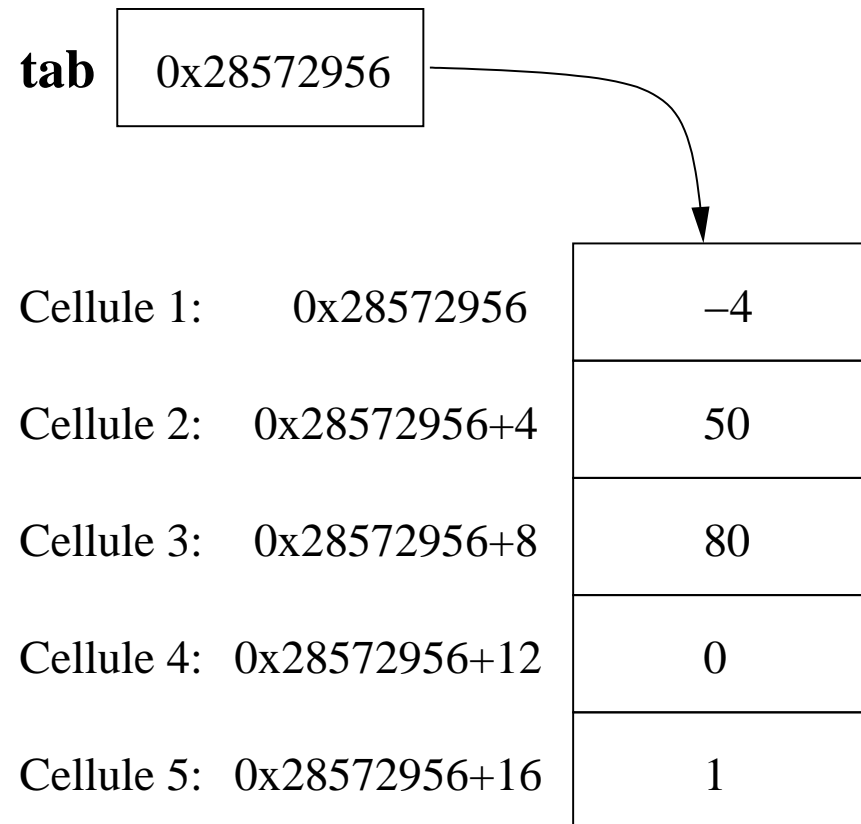
```
Type[] nom = {val1, val2, ..., valN};
```

où `val $i$`  ( $i = 1..N$ ) est la valeur initiale de la  $i^e$  cellule. Le tableau résultant aura  $N$  cellules.



# Exemple

```
int[] tab = {-4, 50, 80, 0, 1};
```



- Un tableau peut être considéré comme un *objet*, mais sa composition est *uniforme* et sa taille totale en mémoire *n'est pas* définie par une *classe*.
- L'attribut est à la classe ce que la cellule est au tableau (du point de vue composition). Avec des classes, pour accéder à un attribut, on fait quelque chose comme

`instance.attribut`

Comment accéder à une cellule d'un tableau?

- Un tableau peut être considéré comme un *objet*, mais sa composition est *uniforme* et sa taille totale en mémoire *n'est pas* définie par une *classe*.
- L'attribut est à la classe ce que la cellule est au tableau (du point de vue composition). Avec des classes, pour accéder à un attribut, on fait quelque chose comme

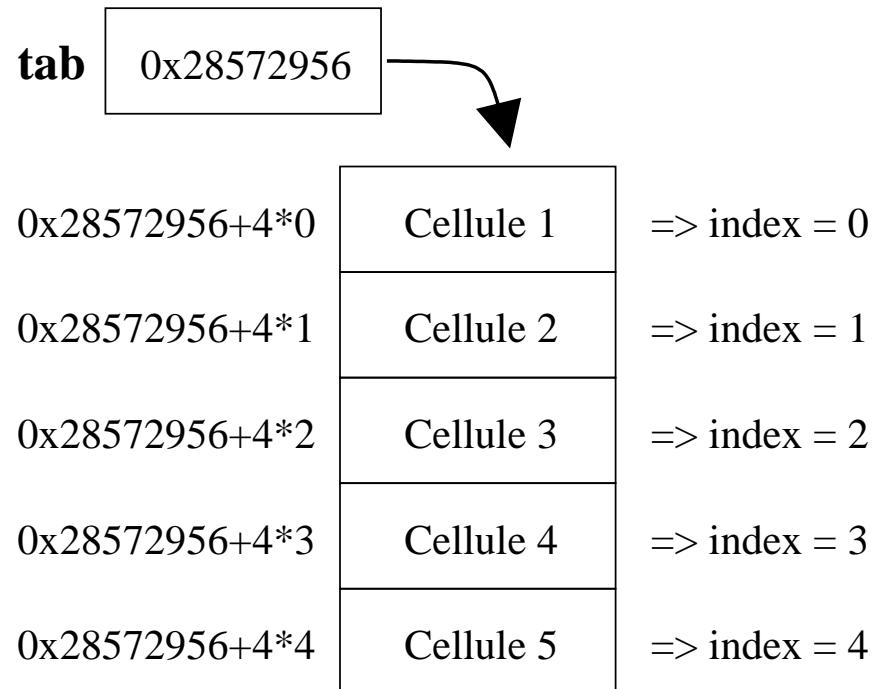
`instance.attribut`

Comment accéder à une cellule d'un tableau?

→ Par indexation.

# Indexation

En reprenant l'exemple du tableau de 5 entiers, on peut réécrire l'adresse de chaque cellule ainsi:



Une indexation naturelle en découle: 1<sup>ère</sup> cellule  $\Rightarrow$  0, 2<sup>e</sup> cellule  $\Rightarrow$  1, ...

# Indexation en Java

- Soit un tableau contenant  $N$  cellules. Pour accéder à la  $i^e$  cellule ( $i = 1..N$ ), la syntaxe en Java est:

`nomDuTableau[i - 1]`

De façon plus générale, on écrit:

`nomDuTableau[index]`

où *index* est une *expression* entière dont l'évaluation donne l'index qu'on désire accéder. Cette valeur *doit* être entre 0 et  $N - 1$ .

- On peut se servir de l'indexation pour accéder à une valeur d'une cellule ou pour y en affecter une autre.

## ***Exemple 1: Lecture de $N$ notes au clavier***

```
// Lecture du nombre de notes à lire.  
System.out.print("Combien de notes? ");  
int n = Keyboard.readInt();  
  
// Instanciation du tableau de notes.  
double[] notes = new double[n];  
  
// Lectures des notes.  
for (int i = 0; i < n; ++i) {  
    System.out.print("Entrez la notes no. " +  
                    (i + 1) + ": ");  
    notes[i] = Keyboard.readDouble();  
}
```

## ***Exemple 2: Calculer la moyenne des $N$ notes***

```
// (fait suite à l'exemple 1)

double moy = 0.0; // la moyenne

// Effectuer d'abord la sommation des notes.
for (int i = 0; i < n; ++i)
    moy += notes[i];

// Calcul final de la moyenne.
moy /= n;
```

# Particularité de Java

- Un tableau en Java est en réalité bel et bien un objet, mais il n'a pas de classe propre à lui.
- Les tableaux en Java ont un attribut: `length`. Soit `tab` une *référence* sur un tableau quelconque, alors

`tab.length`

donne la longueur (i.e. le nombre de cellules) du tableau.

**Note:** Il est impossible de modifier la valeur de cet attribut. Sa valeur est fixée lors de l'instanciation du tableau.

- Comme dans la grande majorité des langages, les tableaux sont passés par références en paramètre des fonctions.



# Exemple

Fonction qui initialise un tableau d'entiers avec une suite contiguë de valeurs commençant par une valeur donnée:

```
public static void iota (int[] tab, int init)
{
    for (int i = 0; i < tab.length; ++i)
        tab[i] = init + i;
}
```

Exemple d'appel:

```
int[] suite = new int[10];
// Ici, suite[i] == 0 pour tout i.
iota(suite, 0);
// Maintenant, suite[i] == i pour tout i.
```

# Bornes des tableaux

- Soit un tableau `t` de longueur  $N$ , les index valides sont entre 0 et  $N - 1$  inclusivement (ou bien `t.length - 1`).
- Que se passe-t-il si on fait `t[i]`, où  $i$  est  $< 0$  ou  $\geq N$ ?

# Bornes des tableaux

- Soit un tableau `t` de longueur  $N$ , les index valides sont entre 0 et  $N - 1$  inclusivement (ou bien `t.length - 1`).

- Que se passe-t-il si on fait `t[i]`, où  $i$  est  $< 0$  ou  $\geq N$ ?

→ Le programme plante!!

- En Java, il y a `ArrayIndexOutOfBoundsException`.

- **Attention:** dans quelques langages, notamment C/C++, il n'y a pas de vérification de bornes.

⇒ Même si le langage effectue une vérification de bornes, il faut être rigoureux!

# Limite des tableaux

---

- Une fois qu'un tableau a été instancié, *on ne peut plus changer sa longueur.*
- ⇒ Il faut connaître le nombre d'éléments à manipuler.

# Tableaux d'objets

- **Rappel:** le type de chaque cellule d'un tableau peut être n'importe quoi.

⇒ peut aussi être un objet.

- Cependant, en Java, un objet est toujours manipulé avec une *référence*.

⇒ un tableau d'objets est en fait un tableau de références sur des objets.

- **Rappel:** dans un tel cas, lors de l'instanciation du tableau, toutes les références sont initialisées à `null`.

⇒ Ne pas oublier de *créer les objets* pour chaque cellule!

Lecture de  $N$  colis postaux:

```
ColisPostal[] colis = new ColisPostal[N];  
for (int i = 0; i < colis.length; ++i)  
    colis[i].lire();
```

⇒ **Ça plante!** car `colis[i] == null` pour tout  $i$ .

Il faut plutôt faire ceci:

```
// instantiation du tableau
ColisPostal[] colis = new ColisPostal[N];

// instantiation de chaque colis
for (int i = 0; i < colis.length; ++i)
    colis[i] = new ColisPostal();

// Enfin, la lecture
for (int i = 0; i < colis.length; ++i)
    colis[i].lire();
```

# Tableaux à deux dimensions

- **Rappel:** le type de chaque cellule d'un tableau peut être n'importe quoi.

⇒ peut aussi être un tableau.

⇒ Pour déclarer un tableau à deux dimensions:

```
Type [] [] nom;
```

i.e. on déclare un *tableau de tableaux*.

- Cependant, l'instanciation est plus particulière:

```
nom = Type [N1] [N2];
```

où  $N_1$  est la longueur de la première dimension (i.e. combien de sous-tableaux) et  $N_2$  est la longueur de la seconde dimension



(i.e. de chaque sous-tableau).

---

# Tableaux à deux dimensions

Soit `Type[][] tab = new Type[N1][N2];`

- Alors `tab[i]` ( $i = 1..N_1$ ) permet d'accéder au sous-tableau à l'index  $i$ .

⇒ Pour accéder à la cellule  $(i, j)$  ( $j = 1..N_2$ ), il suffit d'écrire:

`tab[i][j]`

- On peut aussi connaître la longueur d'un sous-tableau à l'index  $i$  ainsi:

`tab[i].length`

Cependant, ils sont tous de même longueur.

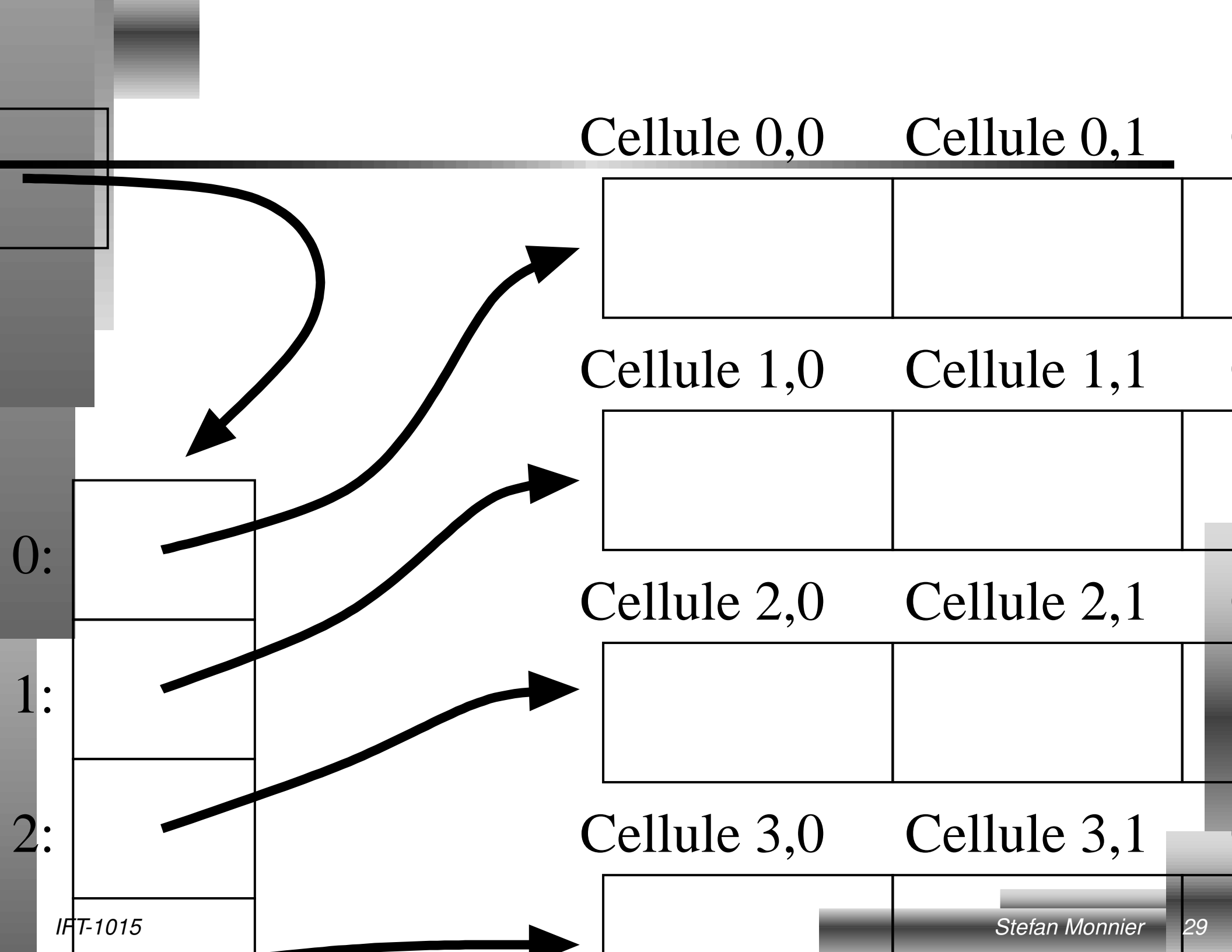
**Remarque:** Il est possible de créer un tableau à deux dimensions où les

tableaux de la seconde dimension ont une longueur différente. Consultez un livre de référence Java pour savoir comment.

## ***Exemple***

Déclaration et instanciation:

```
int[][] tab = new int[5][3];
```



Initialisation:

```
int i, j;
// première dimensions
for (i = 0; i < tab.length; ++i)
{
    // seconde dimensions
    for (j = 0; j < tab[i].length; ++j)
    {
        // Attention à ne pas se tromper dans l'ordre des
        tab[i][j] = (i + j) % 2; // une valeur quelconque
    }
}
```

**Exercice:** Écrivez un bout de code pour afficher ce tableau dans un rectangle  $5 \times 3$ .

# Tableaux à $N$ dimensions

On peut directement généraliser pour avoir un tableau à  $N$  dimensions, i.e. des tableaux de tableaux de tableaux de tableaux de... autant fois que  $N$ .

- Déclaration: mettre autant de paire de crochets qu'il y a de dimensions.
- Instanciation: ajouter un nombre entre crochets pour chaque dimension.
- Indexation: ajouter un index entre crochets pour chaque dimensions.

La  $1^{re}$  paire de crochets correspond à la  $1^{re}$  dimensions.



La 2<sup>e</sup> paire correspond à la 2<sup>e</sup> dimensions.

---

...

## Déclaration

```
Type[] nom;
```

où `Type` est *n'importe quoi* (types de base, classes, tableaux).

## Déclaration initialisée

```
Type[] nom = { val_1, ..., val_N};
```

où  $N$  est la longueur du tableau, et `vali` est la valeur de la  $i^e$  cellule.

## Instanciation

```
nom = new Type[combien];
```

où `combien` est le nombre de cellules.

⇒ `nom.length` sera égal à `combien`.

## Indexation

```
nom[index]
```

où `index` est un entier entre 0 et `nom.length`.