Programmation et Algorithmique II

Ch.3 – Tableaux

Bruno Quoitin
(bruno.quoitin@umons.ac.be)

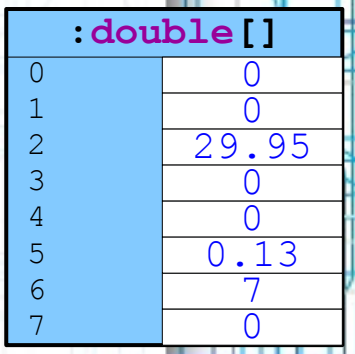
Objectifs

- Les objectifs de ce chapitre sont
  - Introduire le **type tableau** en Java
    - Déclarer un tableau
    - Accéder aux éléments d'un tableau
    - Contraintes d'utilisation : taille fixe, vérification de type, vérification des bornes
  - Tableaux à plusieurs dimensions
  - Copie de tableaux

Table des Matières

Tableaux

- **Type tableau**
  - **Tableau** = séquence d'éléments de même type.
  - Soit `T`, un type qui peut être
    - *primitif* (**`int`**, **`double`**, …)
    - *objet* (`String`, `Carre`, ...)
  - `T[]` désigne le **type tableau** d'éléments de type `T`
    - Exemples
      - **`int[]`** est un type "*tableau d'entiers*"
      - `String[]` est un type "*tableau de* `String`".
  - Deux types tableaux `X[]` et `Y[]` sont compatibles ssi `X` et `Y` sont compatibles (en particulier si `X` = `Y`).



Note : la longueur du tableau ne fait pas partie du type

Tableaux

- **Déclaration d'une variable tableau**
  - La **déclaration d'une variable de type tableau** comporte les éléments suivants
    - le **nom** de la variable tableau (identifiant)
    - le **type** T des éléments du tableau
    - une **initialisation** optionnelle du contenu
  - Syntaxe

(1) *nomType* `[]` *nomVariable* *[* `=` `{` *valeur1, ..., valeurN* `}` *]* `;`

(2) *nomType* *nomVariable* `[]` *[* `=` `{` *valeur1, ..., valeurN* `}` *]* `;`

Dans ce cours, nous préférons la syntaxe (1) car le type `T[]` apparaît.

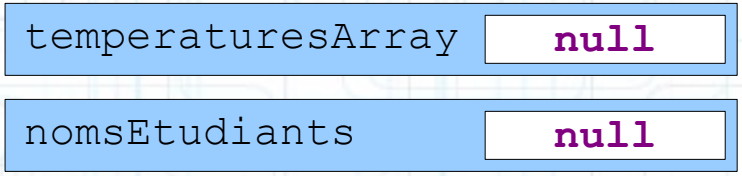
  - Convention de nommage
    - Généralement, les variables tableaux sont **au pluriel**.
    - Elles peuvent également comporter des termes tels que "*Tableau*" ("*Array*") ou "*Elements*" ("*Items*")

Tableaux

- **Déclaration d'une variable tableau**
  - Exemple

```
double[] temperaturesArray;
String[] nomsEtudiants;
```

  - Attention ! Comme pour les objets, une variable de type tableau est un emplacement mémoire contenant une référence vers un tableau.



  - Une étape de création des tableaux est encore nécessaire.

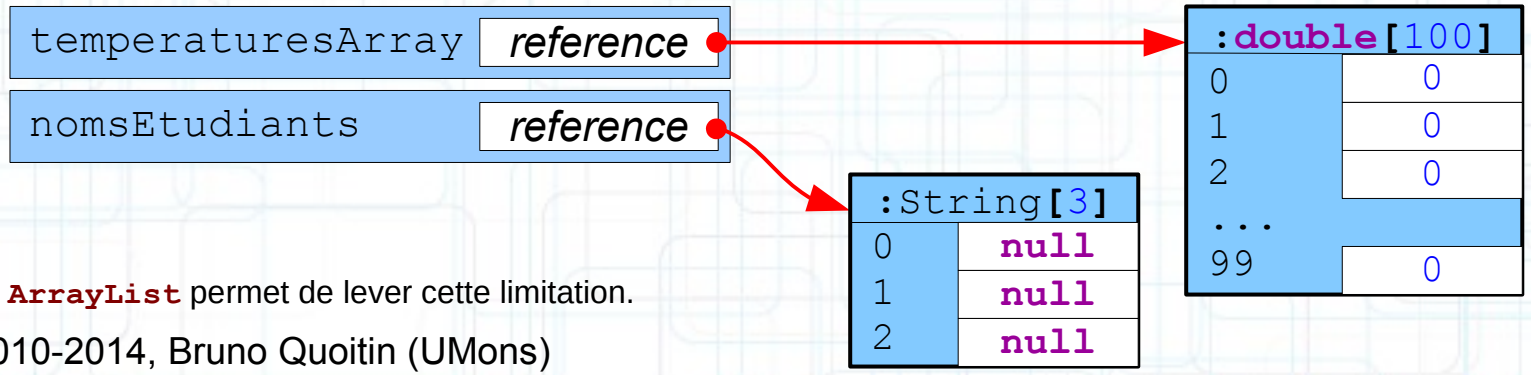
Tableaux

- **Construction d'un tableau**
  - La **construction d'un tableau** est assurée par l'opérateur **new** auquel sont spécifiés le type et la longueur du tableau. La longueur ne peut être changée par la suite(1).
  - Syntaxe

```
new nomType [ nombreElements ]
```

  - Exemple

```
double[] temperaturesArray= new double[100];
String[] nomsEtudiants= new String[3];
```



(1) La classe **ArrayList** permet de lever cette limitation.

Tableaux

- **Construction d'un tableau**
  - Lorsqu'un tableau vient d'être créé, la valeur de chacun de ses éléments est automatiquement initialisée.
  - La valeur assignée à chaque élément du tableau dépend du type de ceux-ci (idem constructeurs par défaut). Le tableau ci-dessous indique les valeurs assignées.

Type des éléments	Valeur initiale
`byte`, `short`, `int`, `long`, `float`, `double`	`0`
`boolean`	`false`
`char`	`'\u0000'`
référence vers objet	`null`

Tableaux

- **Initialisation d'un tableau**
  - Il est possible d'**allouer et initialiser les cellules** d'un tableau lors de sa création. Le type des valeurs initiales doit être compatible avec le type du tableau.
  - Syntaxe

```
nomType [] nomVariable = { valeur1 , ... , valeurN } ;
```

  - Exemple

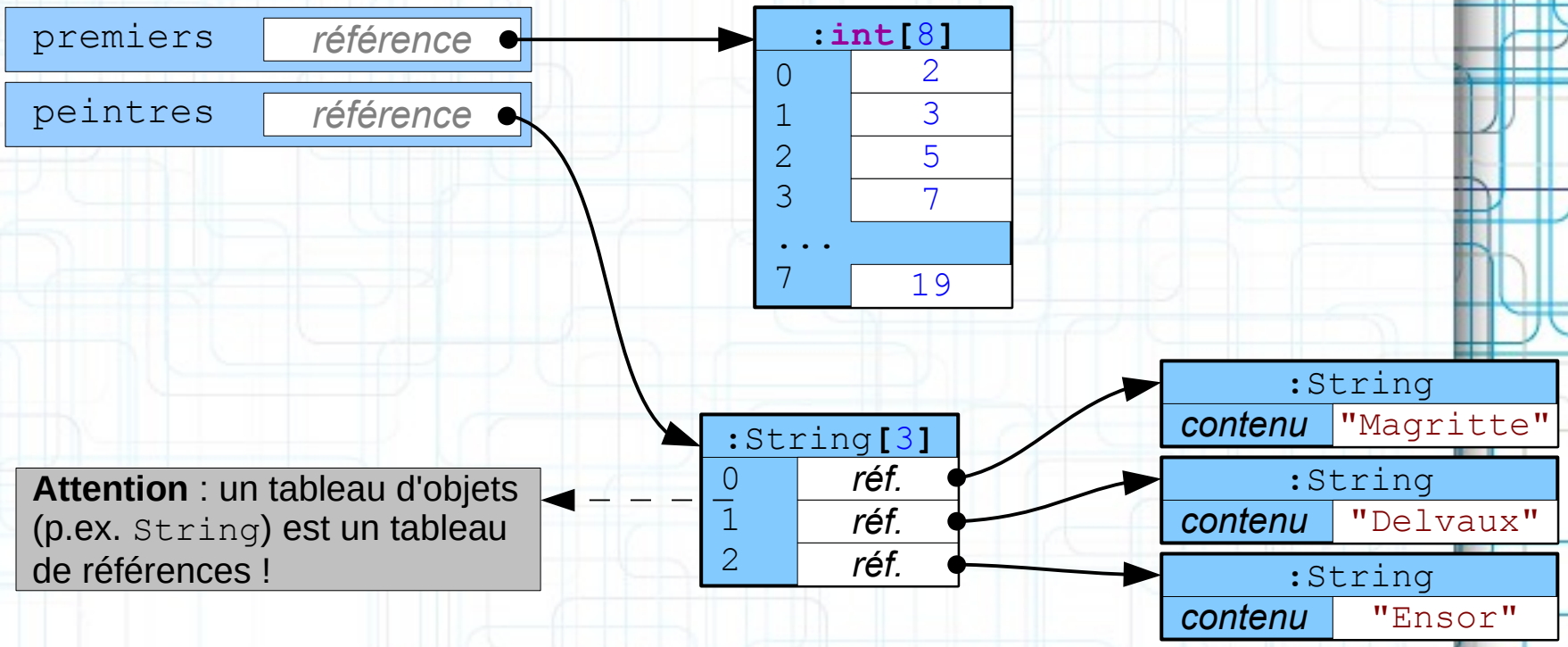
```
int[] premiers= { 2, 3, 5, 7, 11, 13, 17, 19 };
String[] peintres= { "Magritte", "Delvaux", "Ensor" };
```

  - **Note :** bien que l'opérateur **new** ne soit pas appelé explicitement, il y a allocation d'un objet tableau. Les valeurs fournies entre les accolades sont copiées dans les cellules du tableau. La taille du tableau correspond au nombre de valeurs spécifiées.

Tableaux

- **Initialisation d'un tableau**
  - Exemple

```
int[] premiers= { 2, 3, 5, 7, 11, 13, 17, 19 };
String[] peintres= { "Magritte", "Delvaux", "Ensor" };
```



Tableaux

- **Nombre d'éléments**
  - Le **nombre d'éléments** (longueur) d'un tableau est accessible au travers d'un attribut nommé `length`.
    - `length` est une variable d'instance.
    - `length` est déclarée avec le mot-clé **`final`** ⇒ sa valeur est assignée une fois pour toutes par le constructeur du tableau (rappel : un tableau a une longueur fixe).
  - Exemple

```
double [] temperaturesArray= new double[100];
String[] peintres= { "Magritte", "Delvaux", "Ensor" };
System.out.println(temperaturesArray.length);
System.out.println(peintres.length);
```

Tableaux

Note : il n'existe pas en Java d'opérateur de *slicing* comme en Python. En revanche, il est possible d'utiliser les méthodes `CopyOfRange` de la classe `java.util.Arrays`

- **Indexation d'un tableau**
  - L'**opérateur d'indexation** permet d'accéder aux éléments d'un tableau. C'est un opérateur binaire qui prend la référence d'un tableau et un index entier.
    - L'index est une expression entière dont la valeur doit être comprise entre 0 et `length`-1.
    - L'élément résultant peut être utilisé dans une affectation (*left-value*) ou dans une expression (*right-value*).
  - Syntaxe

```
referenceTableau [ index ]
```

  - Exemple

```
String msg= temperatures[5] < 3 ? "Danger gel" : "";
nomsEtudiants[2]= "Bill";
```

Tableaux

- **Indexation d'un tableau**
  - Lors de l'exécution, la JVM vérifie que tout accès à un tableau se situe **entre les bornes**
    - borne inférieure = 0
    - borne supérieure = `length`-1.
    - En cas d'accès en dehors des bornes, une exception `ArrayIndexOutOfBoundsException` est générée.
  - Exemple

```
int[] array= new int[10];
array[10]= 5;          /* accès en dehors des bornes */
```

```
bash-3.2$ java MonProgramme
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at MonProgramme.main(MonProgramme.java:6)
bash-3.2$
```

Tableaux

- **Utilisation d'un tableau**
  - Exemple

```
final int NUM_TEMPERATURES = 100;
double[] temperatures= new double[NUM_TEMPERATURES];
temperatures[0]= 23.4;
temperatures[1]= 22.6;
temperatures[3]= 19.2;
...
int index= 0;
double moyenne= 0;
while (index < temperatures.length) {
    moyenne= moyenne + temperatures[index];
    index++;
}
moyenne= moyenne / temperature.length;
```

Bonne pratique : utiliser une constante nommée plutôt qu'une « valeur magique ».

Utilisation comme *left-value* (dans une affectation)

Utilisation comme *right-value* (dans une expression)

Tableaux

- **Boucle *for-each***
  - La boucle **`for-each`**[1] (ou *enhanced-*`for`) permet de parcourir séquentiellement l'ensemble des éléments d'un tableau[2]. Elle permet de rendre le code plus compact et plus lisible.
  - Syntaxe

```
for ( nomType nomVariable : tableau )
    blocInstructions
```

  - Exemple

```
final int LENGTH = 100;
char[] tab= new char[LENGTH];
/* … */
String msg= "";
for (int i= 0; i < tab.length; i++)
    msg = msg + tab[i];
```

⇨

```
final int LENGTH = 100;
char[] tab= new char[LENGTH];
/* … */
String msg= "";
for (char c : tab)
    msg = msg + c;
```

(1) disponible depuis Java 5.0.
(2) elle s'applique aussi aux `Collections` (voir plus loin dans le cours).

Tableaux

- **Méthodes utilitaires**
  - La bibliothèque java a été peu à peu enrichie de nombreuses méthodes de classe permettant d'effectuer des opérations utiles sur les tableaux (copie, tri, comparaison, test d'égalité, recherche dichotomique, etc).
  - Ces méthodes sont fournies par la classe `java.util.Arrays`

```
static int binarySearch(T[] a, T key);
static int compare(T[] a, T[] b);
static T[] copyOf(T[] a, int newLength);
static T[] copyOfRange(T[] a, int from, int to);
static boolean equals(T[] a, T[] b);
static void fill(T[] a, T val);
static int mismatch(T[] a, T[] b);
static void sort(T[]);
static String toString(T[] a);
static boolean deepEquals(T[] a, T[] b);
static String deepToString(T[]);
/* et bien d'autres … */
```

ATTENTION. Dans ce contexte, le type T dénote l'un des types primitifs, le type Object ou un type générique.

Des variantes de ces méthodes sont fournies par surcharge pour chaque possibilité de T.

Tableaux

- Equivalent des accès aux listes en Python
  - indexation en partant de la fin ($1 \leq i \leq N$)

```
Python:  a[-i]
Java:    a[a.length - i]
```

  - tranche de *start* jusqu'à *stop*-1 ($0 \leq start < stop \leq N$)

```
Python:  a[start:stop]
Java:    Arrays.copyOfRange(a, start, stop)
```

  - tranche depuis *start* ($0 \leq start < N$)

```
Python:  a[start:]
Java:    Arrays.copyOfRange(a, start, a.length)
```

  - tranche jusqu'à *stop*-1 ($0 < stop \leq N$)

```
Python:  a[:stop]
Java:    Arrays.copyOfRange(a, 0, stop)
```

  - copie

```
Python:  a[:]
Java:    Arrays.copyOf(a)
```

Table des Matières

Tableaux

- **Arguments du programme**
  - Les arguments passés en ligne de commande à un programme Java sont passés dans un tableau de `String` à la méthode `main`.
  - Exemple

```
public static void main(String[] args) {
    int index= 0;
    while (index < args.length) {
        System.out.println("Arg["+index+"]: \""+args[index]+"\"");
        index++;
    }
}
```

```
bash-3.2$ java ArgsExample 5 "coco est content"
Arg[0]: "5"
Arg[1]: "coco est content"
bash-3.2$
```

Tableaux

- **Méthode à nombre variable d'arguments**
  - Il est possible de définir une méthode prenant un **nombre variable d'arguments** (*vararg*). Ces arguments sont passés à la méthode sous forme d'un tableau.
  - Exemple

```
public static void methode(int param1, String... varargs) {
    System.out.println("Param 1 = "+param1);
    for (int index= 0; index < varargs.length; index++)
        System.out.println("Param "+(index+1)+" = "+varargs[index]);
}
```

Les éventuels arguments obligatoires précèdent l'argument de taille variable (*vararg*).

La déclaration de l'argument de taille variable (*vararg*) doit toujours être la dernière de la liste et ne peut apparaître qu'une fois.

Tableaux

- **Méthode à nombre variable d'arguments**
  - Exemple (suite)

```
public static void methode(int param1, String... varargs) {
    System.out.println("Param 1 = "+param1);
    for (int index= 0; index < varargs.length; index++)
        System.out.println("Param "+(index+2)+" = "+varargs[index]);
}
```

```
methode(1, "coucou", "coco");
```

```
Param 1 = 1
Param 2 = coucou
Param 3 = coco
```

```
methode(2);
```

```
Param 1 = 2
```

```
methode(3, "salut", 1234);
```

Ne sera pas accepté par le compilateur : il n'est pas possible de mettre un entier dans un tableau de `String`.

Tableaux

- **Cas de la méthode `printf`**
  - `printf` écrit sur un flux de sortie (p.ex. la console). Le format d'affichage est spécifié à l'aide d'une *chaîne de caractère de format*.

```
public void printf(String format, Object... args);
```

  - La chaîne `format` spécifie comment afficher chacun des éléments de `args` selon le format suivant

`%` [*flags*] [*largeur*] [*.precision*] *conversion*

  - Exemple

```
System.out.printf("%04X\n", 165);              /* 00A5 */
System.out.println(Math.PI);    /* 3.141592653589793 */
System.out.printf("%.2f\n", Math.PI);         /* 3.14 */
System.out.printf("Date = %02d/%02d/%04d\n",
                  jour, mois, annee); /* 30/01/2017 */
```

X = entier, hexadécimal
4 = largeur
0 (*flags*) = *"zero-padding"*

f = flottant
.2 = chiffres après virgule

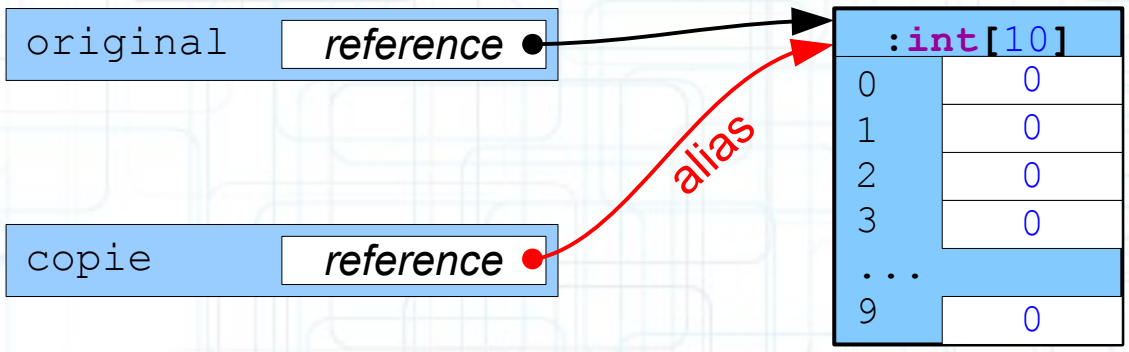
d : entier, décimal
2 = largeur
0 = zero-padding

Table des Matières

Tableaux

- **Duplication de tableaux**
  - Il est parfois également nécessaire de dupliquer un tableau, i.e. créer un nouveau tableau dont la taille et le contenu sont identiques à un autre tableau.
  - On ne peut se limiter à créer une nouvelle variable tableau et lui affecter la valeur de l'ancienne variable tableau → **cela crée un alias !!!**
  - Exemple incorrect

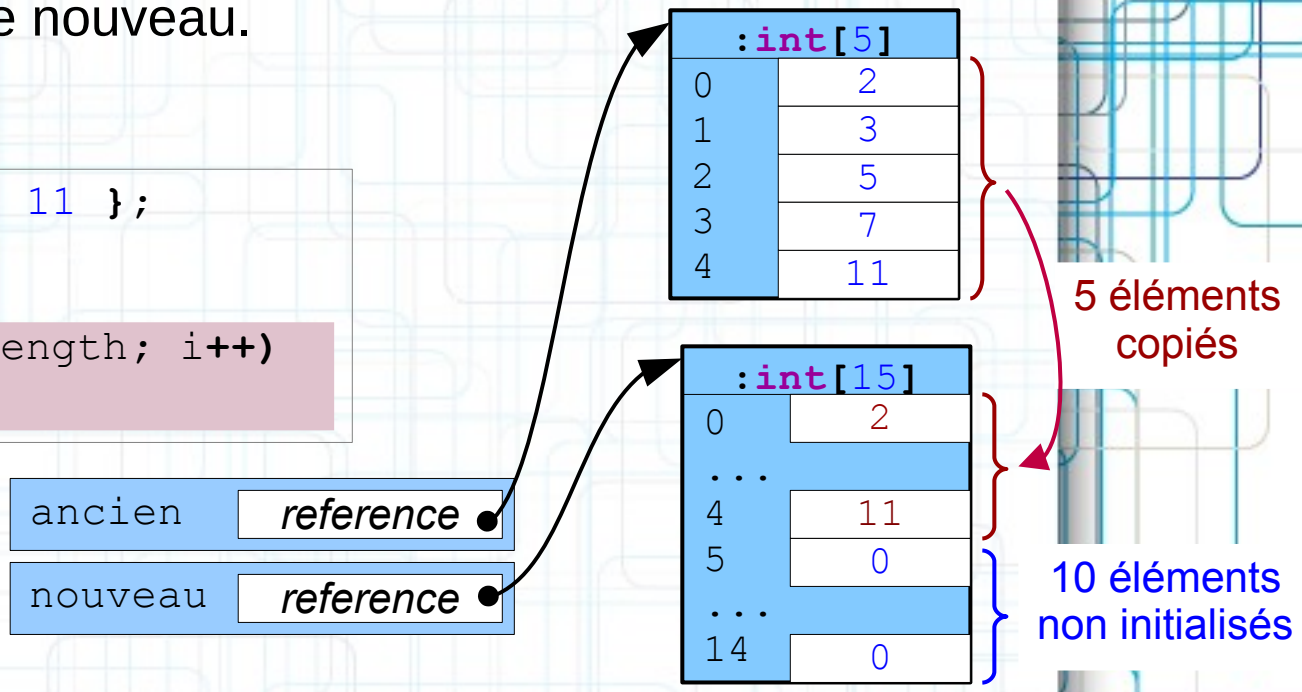
```
int[] original= new int[10];
int[] copie= original;
```



Tableaux

- **Copie de tableaux**
  - Les tableaux ont le désavantage d'avoir un nombre d'éléments fixé lors de leur construction → Il n'est **pas possible d'augmenter ou de réduire la taille d'un tableau**.
  - Lorsque la taille d'un tableau doit être changée, il est nécessaire d'**allouer un nouveau tableau et de copier** les éléments de l'ancien tableau vers le nouveau.
  - Exemple

```
int[] ancien= { 2, 3, 5, 7, 11 };
/* … */
int[] nouveau= new int[15];
for (int i= 0; i < ancien.length; i++)
  nouveau[i]= ancien[i];
```



Tableaux

- **Copie de tableaux**
  - La bibliothèque Java fournit des méthodes spécifiques destinées à la copie de tableaux. Un exemple est la méthode arraycopy qui permet de remplacer la copie « à la main » montrée au *slide* précédent.

```
static void arraycopy(Object from,
                      int fromIndex,
                      Object to,
                      int toIndex,
                      int count);
```

  - Exemple

```
int[] ancien= { 2, 3, 5, 7, 11 };
/* … */
int[] nouveau= new int[15];
System.arraycopy(ancien, 0, nouveau, 0, ancien.length);
```

Méthode de classe (**static**) de la classe `System`.

Tableaux

- **Concaténation de deux tableaux**
  - Une autre opération parfois nécessaire est la concaténation de deux tableaux. Il n'y a pas en Java d'opérateur permettant cette concaténation.
  - Exemple

```
int[] tab1 = { 2, 3, 5, 7, 11 };
int[] tab2 = { 13, 17, 19, 23 };
int[] concat = new int[tab1.length + tab2.length];
System.arraycopy(tab1, 0, concat, 0, tab1.length);
System.arraycopy(tab2, 0, concat, tab1.length, tab2.length);
```

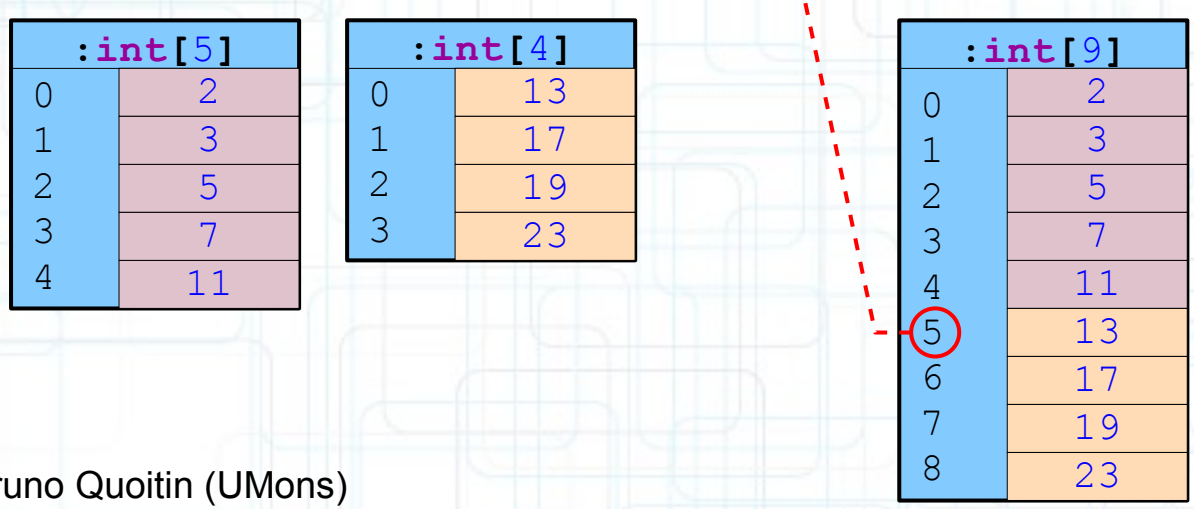


Table des Matières

Tableaux

- **Exercice**
  - Tester qu'un tableau est trié (croissant, non strictement). Pré-condition : tableau non vide.

```
import java.util.Arrays;

public class TestArray {

  public static boolean isArraySorted(int[] a) { }
    /* implémentation à fournir */

  public static void main(String [] args) {
    int[] a1= { 1, 3, 3, 7, 9 };
    int[] a2= { -8, 9, 11, 9, 8, 9 };

    System.out.println(Arrays.toString(a1) +
      " = " + isArraySorted(a1));
    System.out.println(Arrays.toString(a2) +
      " = " + isArraySorted(a2));
  }
}
```

Tableaux

- **Exercice**
  - Déterminer les valeurs minimum et maximum d'un tableau. Pré-condition : tableau non vide.

```
import java.util.Arrays;

public class TestArray {

  public static int getMinimum(int[] a) { }
    /* implémentation à fournir */

  public static int getMaximum(int[] a) { }
    /* implémentation à fournir */

  public static void main(String [] args) {
    int[] a= { 1, 3, 3, 7, 9 };

    System.out.print(Arrays.toString(a));
    System.out.print(", min=" + getMinimum(a));
    System.out.println(", max=" + getMaximum(a));
  }
}
```

Tableaux

- **Exercice**
  - Ecrire une méthode qui permette d'évaluer la valeur d'un polynôme pour une valeur de la variable x. Les coefficients du polynôme sont passés en argument comme *vararg*.

```
public class Polynome {

  public static double eval(double x, double… coeffs) { }
    /* implémentation à fournir */

  public static void main(String [] args) {
    System.out.println(eval(5, 1.5, 3, 1));
    System.out.println(eval(5, 0, 0, 1));
    System.out.println(eval(5, 0, 0, -1, 2));
  }

}
```

$x^2+3x+\frac{3}{2}$

x^2

$2x^3-x^2$

Tableaux

- **Exercice**
  - Soit un tableau contenant des 0 et des 1, tel que le tableau puisse être découpé en 2 parties. Dans la partie de gauche il n'y a que des 0 et dans celle de droite que des 1.

0	0	0	0	0	1	1

  - Il vous est demandé d'écrire une méthode Java `int somme01(int [] tab)` qui calcule la somme des éléments du tableau. Pour le tableau donné en exemple ci-dessus, le résultat de la méthode doit être 2.
  - Il est important que vous conceviez votre méthode de façon à **nécessiter un minimum d'itérations**. Un algorithme qui itère sur l'ensemble des cellules du tableau ne sera pas considéré comme suffisant pour réussir cette question !

Cette question est inspirée du questionnaire de la finale des olympiades informatiques belges, section étudiants du secondaire.

Table des Matières

Tableaux multi-dimensionnels

- **Type, déclaration, construction**
  - `S=T[]` étant un type objet, il est possible de définir un tableau de type `S[]=T[][]`...
  - Syntaxe de la déclaration

```
nomType [ ]...[ ] nomVariable ;
```

Le nombre de paires de crochets détermine le nombre de dimensions du tableau.

  - Syntaxe de la construction

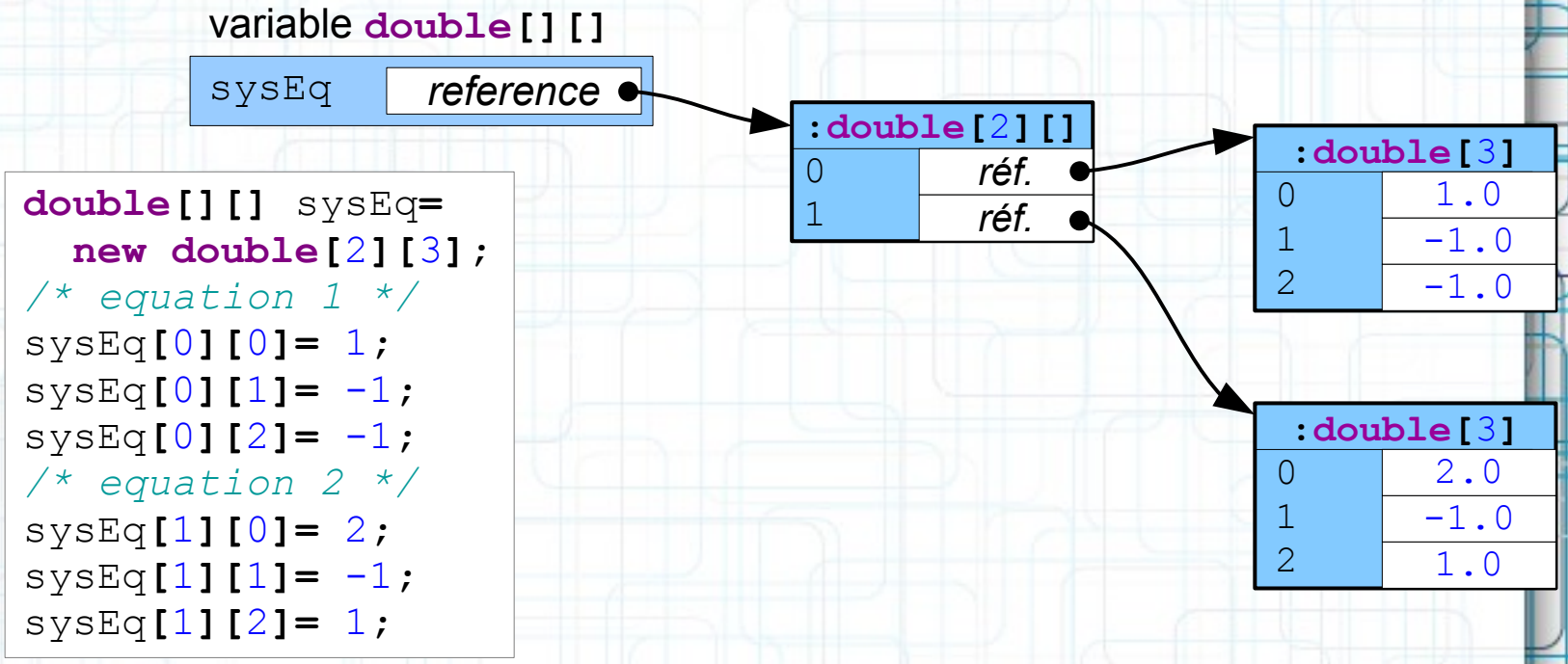
```
new nomType [ nombreEltsDim1 ]...[ nombreEltsDimN ]
```

La taille de chaque dimension est spécifiée lors de la construction.

Note : La spécification de Java ne limite pas le nombre de dimensions. Cependant, le nombre de dimensions est typiquement limité à 255 dans l'implémentation de la JVM.

Tableaux multi-dimensionnels

- **En fait...**
  - ... les tableaux multi-dimensionnels **n'existent pas en Java !!** Il s'agit en réalité de tableaux de tableaux.

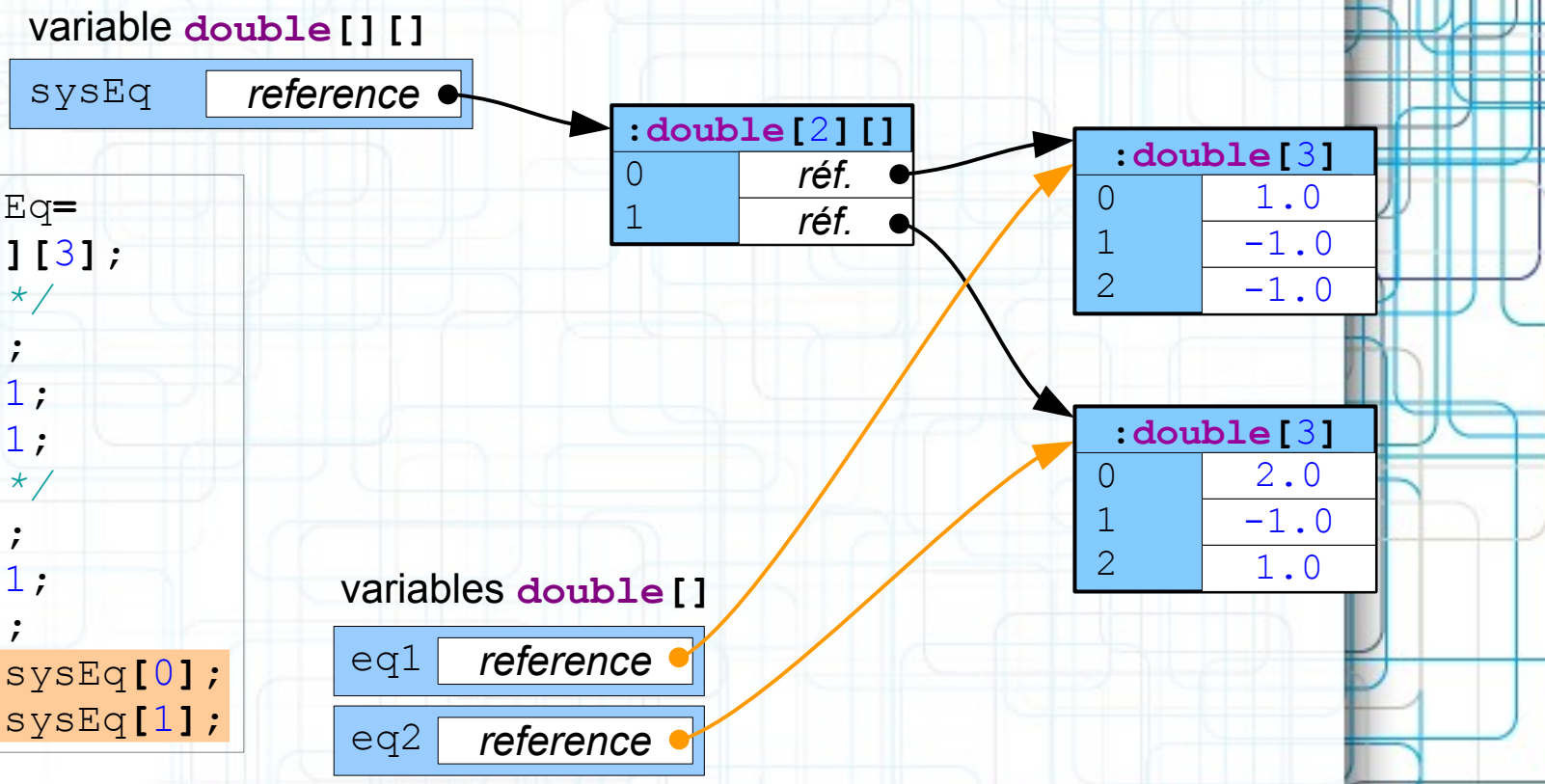



```
double[][] sysEq=
  new double[2][3];
/* equation 1 */
sysEq[0][0]= 1;
sysEq[0][1]= -1;
sysEq[0][2]= -1;
/* equation 2 */
sysEq[1][0]= 2;
sysEq[1][1]= -1;
sysEq[1][2]= 1;
```

Tableaux multi-dimensionnels

- **Références vers les sous-tableaux**
  - Il est possible de manipuler directement les références vers les "sous-tableaux".

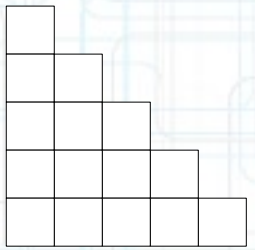
```
double[][] sysEq=
  new double[2][3];
/* equation 1 */
sysEq[0][0]= 1;
sysEq[0][1]= -1;
sysEq[0][2]= -1;
/* equation 2 */
sysEq[1][0]= 2;
sysEq[1][1]= -1;
sysEq[1][2]= 1;
double[] eq1= sysEq[0];
double[] eq2= sysEq[1];
```

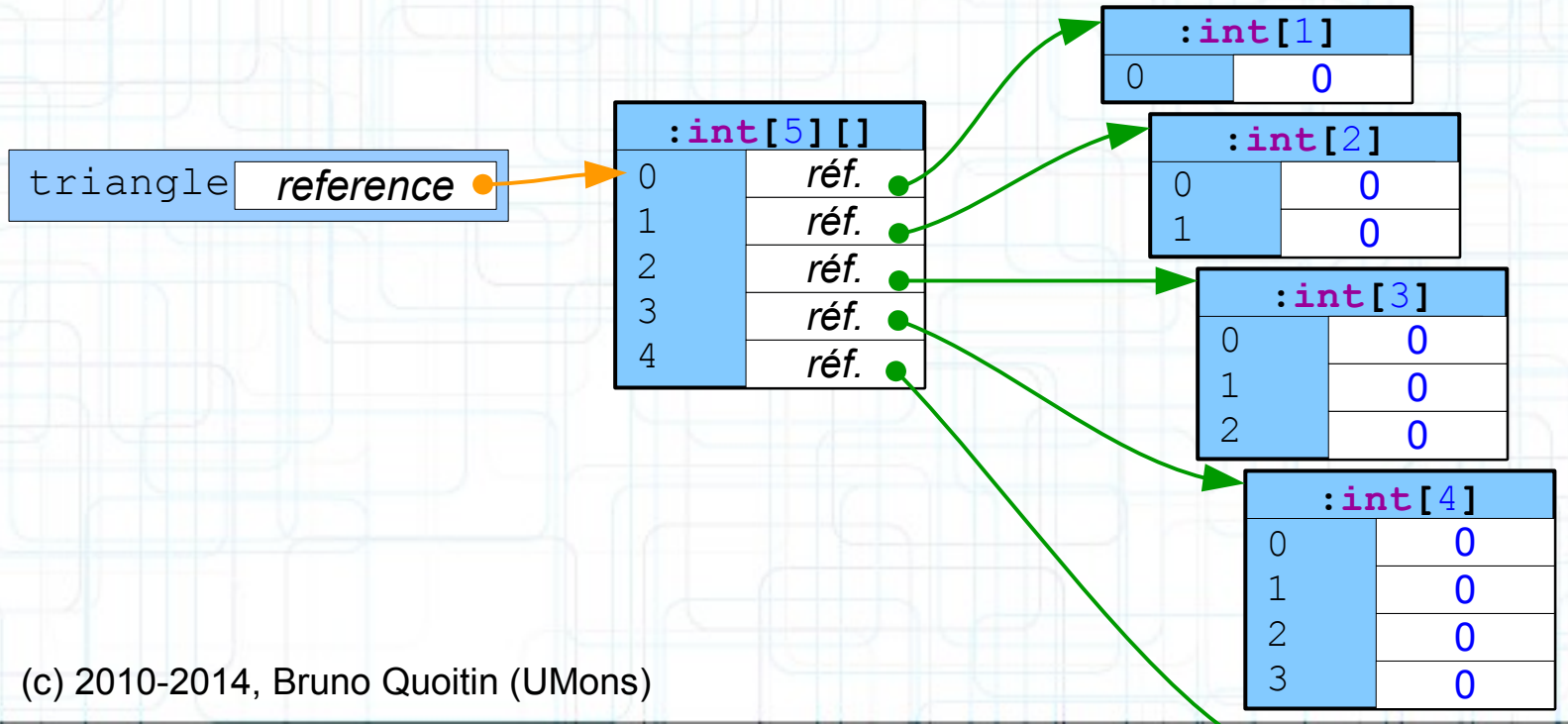


Tableaux multi-dimensionnels

- **Tableaux internes hétérogènes**
  - Les tableaux internes d'un tableau de tableaux peuvent avoir des tailles différentes.

```
int[][] triangle= new int[5][];
for (int i= 0; i < triangle.length; i++)
  triangle[i]= new int[i+1];
```

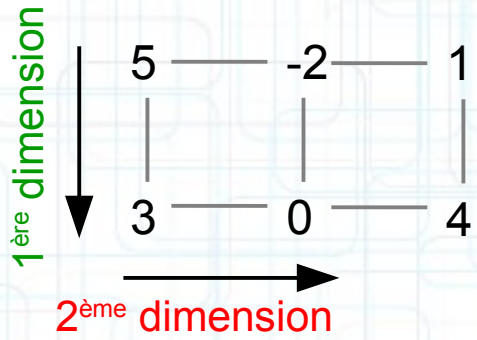




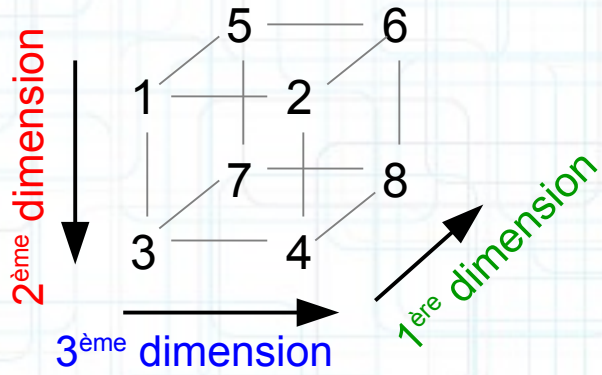
Tableaux multi-dimensionnels

- **Initialisation**
  - Il est possible d'initialiser des tableaux « multi-dimensionnels » lors de leur création. La syntaxe est similaire à celle des tableaux à une dimension.
  - Exemples

```
double[][] matrice2d =
  { { 5, -2, 1},
    { 3, 0, 4 } };
```



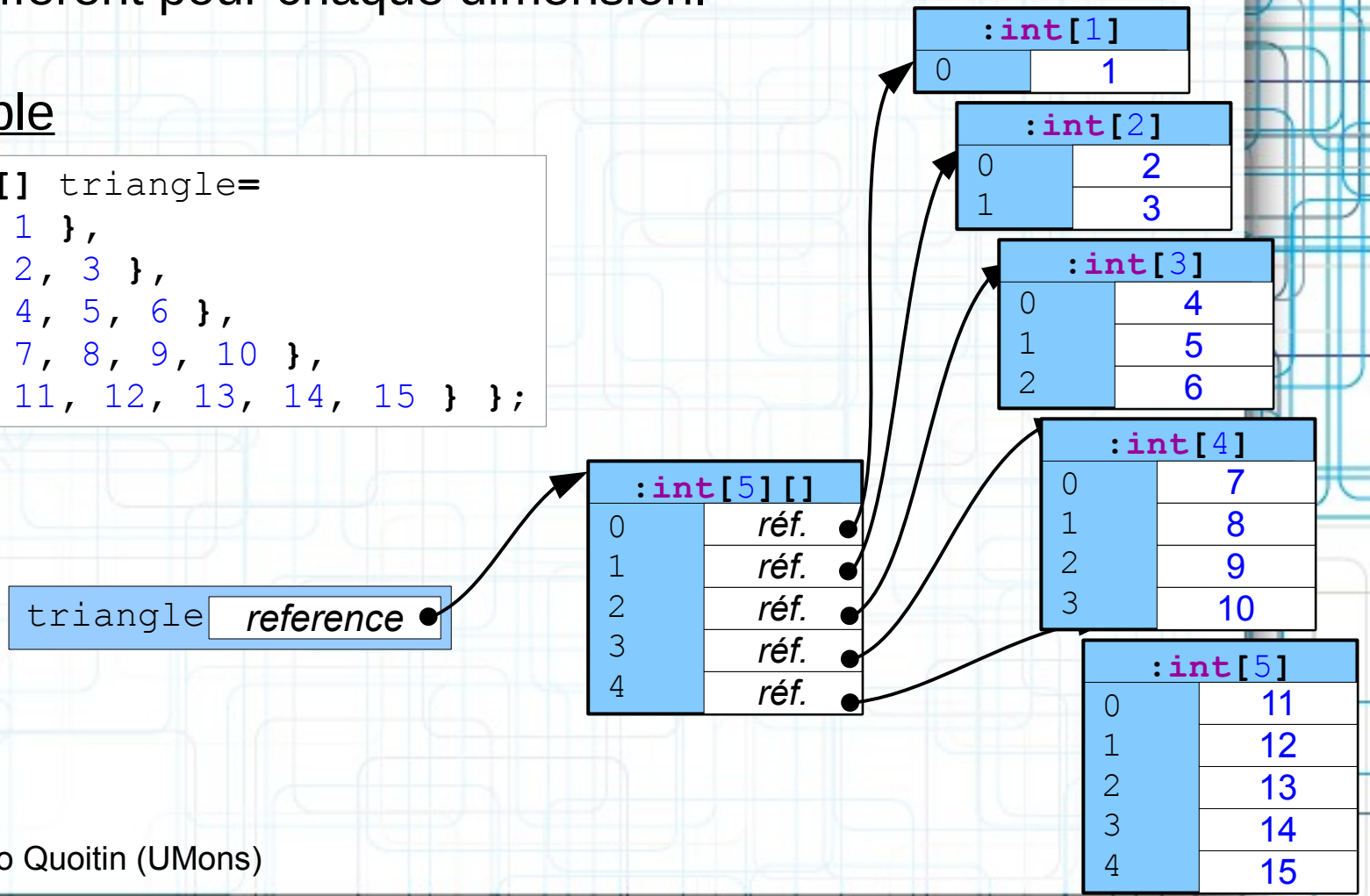

```
int[][][] matrice3d =
  { { { 1, 2 },
      { 3, 4 } },
    { { 5, 6 },
      { 7, 8 } } };
```



Tableaux multi-dimensionnels

- **Initialisation**
  - Le nombre d'éléments fournis lors de l'initialisation peut être différent pour chaque dimension.
  - Exemple

```
int[][] triangle=
   { { 1 },
     { 2, 3 },
     { 4, 5, 6 },
     { 7, 8, 9, 10 },
     { 11, 12, 13, 14, 15 } };
```



Tableaux multi-dimensionnels

- **Exercice – Multiplication de matrices**
  - Donner une méthode de classe `multiplier` permettant d'effectuer la multiplication d'une matrice par une autre.
  - Les matrices sont modélisées par des tableaux à 2 dimensions de **`double`**.

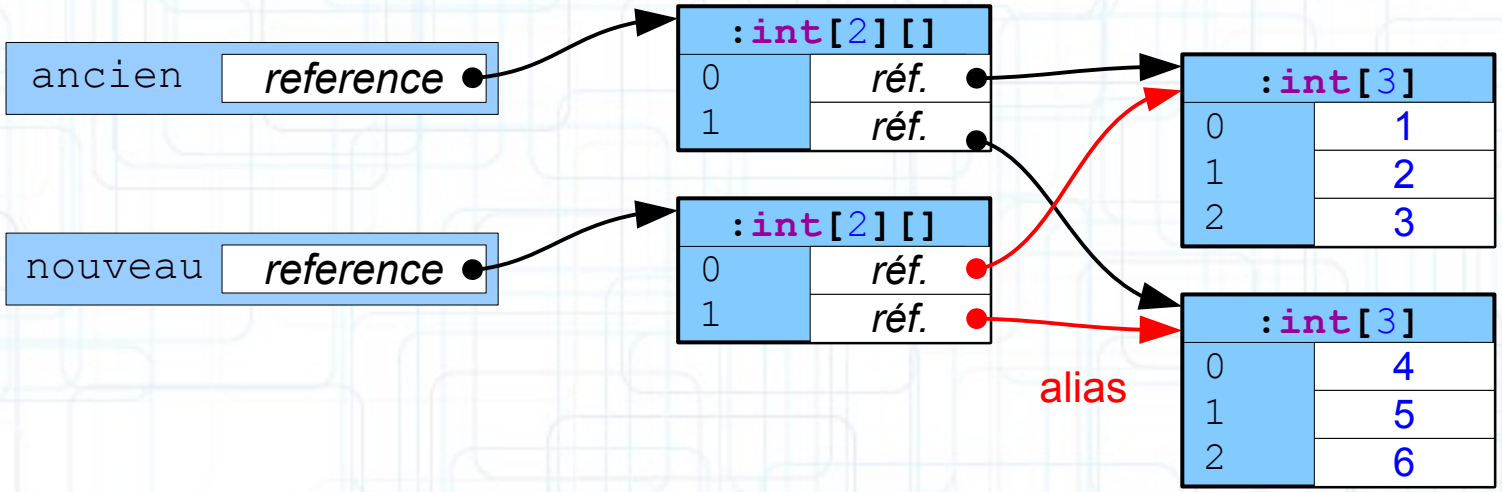
```
public static double[][] multiplier(double[][] a, double[][] b)
{
  if (a[0].length != b.length)
    return null;

  double[][] c = new double [a.length][b[0].length];
  for (int i = 0; i < a.length; i++)
    for (int j = 0; j < b[0].length; j++) {
      double s = 0;
      for (int k = 0; k < b.length; k++)
        s += a[i][k] * b[k][j];
      c[i][j] = s;
    }
  return c;
}
```

Table des Matières

Tableaux multi-dimensionnels

- **Copie**
  - Attention : la méthode `arraycopy` **ne fonctionne pas pour des tableaux multi-dimensionnels**. La méthode copie uniquement le tableau le plus externe.
  - Exemple

```
int[][] ancien= { { 1, 2, 3 }, { 4, 5, 6 } };
int[][] nouveau= new int[2][3];
System.arraycopy(ancien, 0, nouveau, 0, ancien.length);
```



Tableaux multi-dimensionnels

- **Copie**
  - Pour copier des tableaux multi-dimensionnels, il faut boucler sur le tableau de tableau et effectuer la copie de chaque sous-tableau.
  - Exemple

```
int[][] ancien= { { 1, 2, 3 }, { 4, 5, 6 } };
int[][] nouveau= new int[2][3];
for (int i= 0; i < ancien.length; i++)
  System.arraycopy(ancien[i], 0, nouveau[i], 0, ancien[i].length);
```

  - Autres cas à traiter ?
    - s'il y a plus que 2 dimensions ?
    - si les sous-tableaux n'ont pas tous la même taille ?

Tableaux multi-dimensionnels

- **Copie**
  - La copie générique de tableaux de dimensions et types quelconques requiert d'apprendre quelques détails supplémentaires de Java. En particulier sur l'héritage...
  - Idée de solution

```
public static void copy(Object[] src, Object[] dst) {
    for (int i= 0; i < src.length; i++) {
      if (src[i] instanceof Object[])
        copy2((Object[]) src[i], (Object[]) dst[i]); /* cas récursif */
      else if (src[i] instanceof int[])
        System.arraycopy(src[i], 0, dst[i], 0,
            ((int []) src[i]).length);
      else
        /* traiter de la même manière les cas byte[], short[],
           long[], boolean[], char[], double[] et float[] */
      else
        dst[i]= src[i]; /* pas un tableau */
    }
  }
}
```

cloner ?

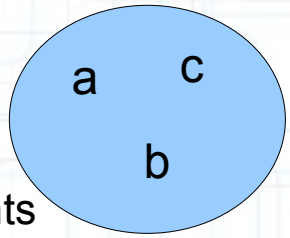
Non testé

Table des Matières

Tableaux multi-dimensionnels

- **Application au calcul de combinaisons**
  - L'objectif est de déterminer le nombre de façons qu'il y a de prendre k éléments (sans replacement) parmi n.
  - Exemple : combien de combinaisons dans les cas suivants
    - n=1, k=1
    - n=2, k=1
    - n=3, k=2

Illustration pour n=3, k=2



Ensemble à $n = 3$ éléments

{a, b}
{a, c}
{b, c}

Parties à $k = 2$ éléments

  - Le nombre de combinaisons de k éléments pris dans un ensemble de n éléments peut-être calculé de la façon suivante

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Tableaux

- **Combinaisons : récurrence**
  - Un autre moyen de calculer le nombre de façons de prendre k éléments parmi n consiste à utiliser la relation de récurrence suivante.

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \quad \forall\, 0 < k < n$$

  - Les cas de base sont

$$\binom{n}{0} = 1 \qquad \binom{n}{n} = 1$$

$$\underline{k = 0} \qquad \underline{k = n}$$

$$\forall\, k > n, \quad \binom{n}{k} = 0$$

Tableaux

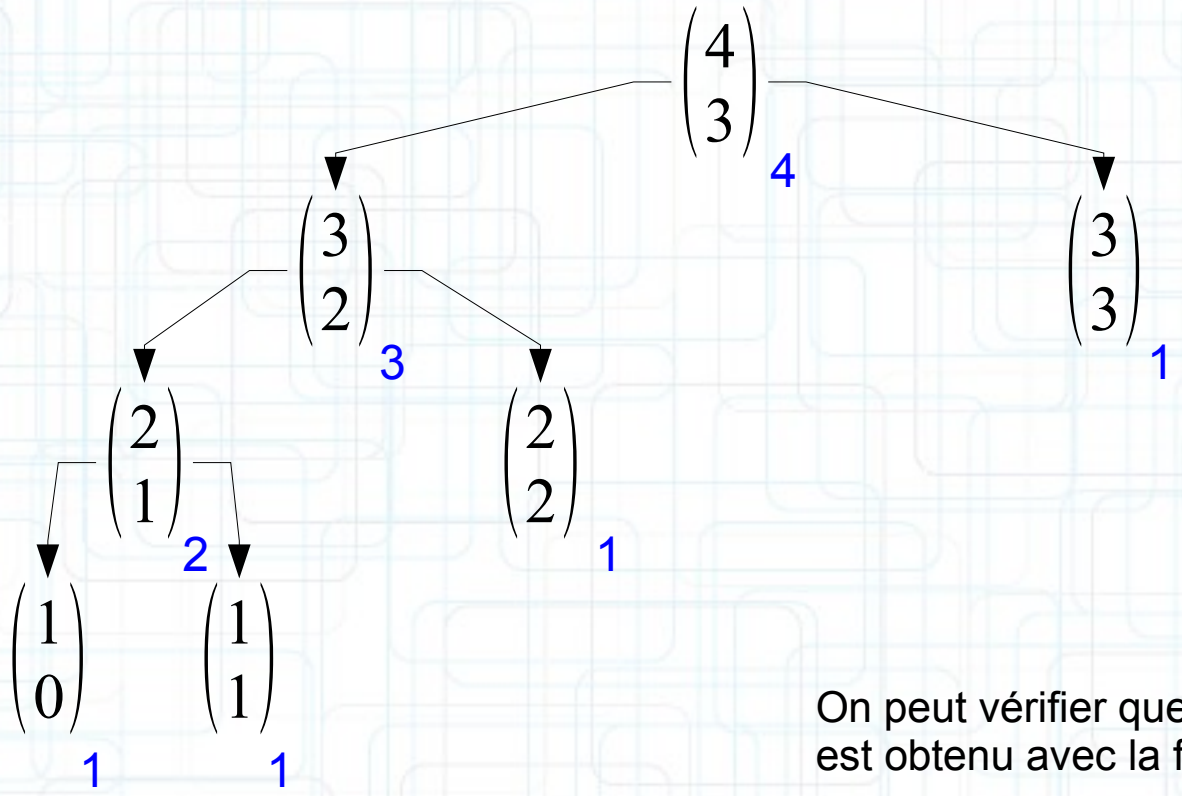
- **Combinaisons : récurrence**
  - Un algorithme récursif vient « immédiatement » à l'esprit.

```
public static int comb(int n, int k)
{
  if ((k == 0) || (k == n))
    return 1;
  if (k > n)
    return 0;
  return comb(n-1, k-1) + comb(n-1, k);
}
```

  - Il s'agit d'une double récursion. Le problème de cette approche est qu'elle peut nécessiter un très grand nombre d'appels récursifs.

Tableaux

- **Combinaisons : récurrence**
  - Exemple : calcul du nombre de façons de prendre k=3 éléments parmi n=4.

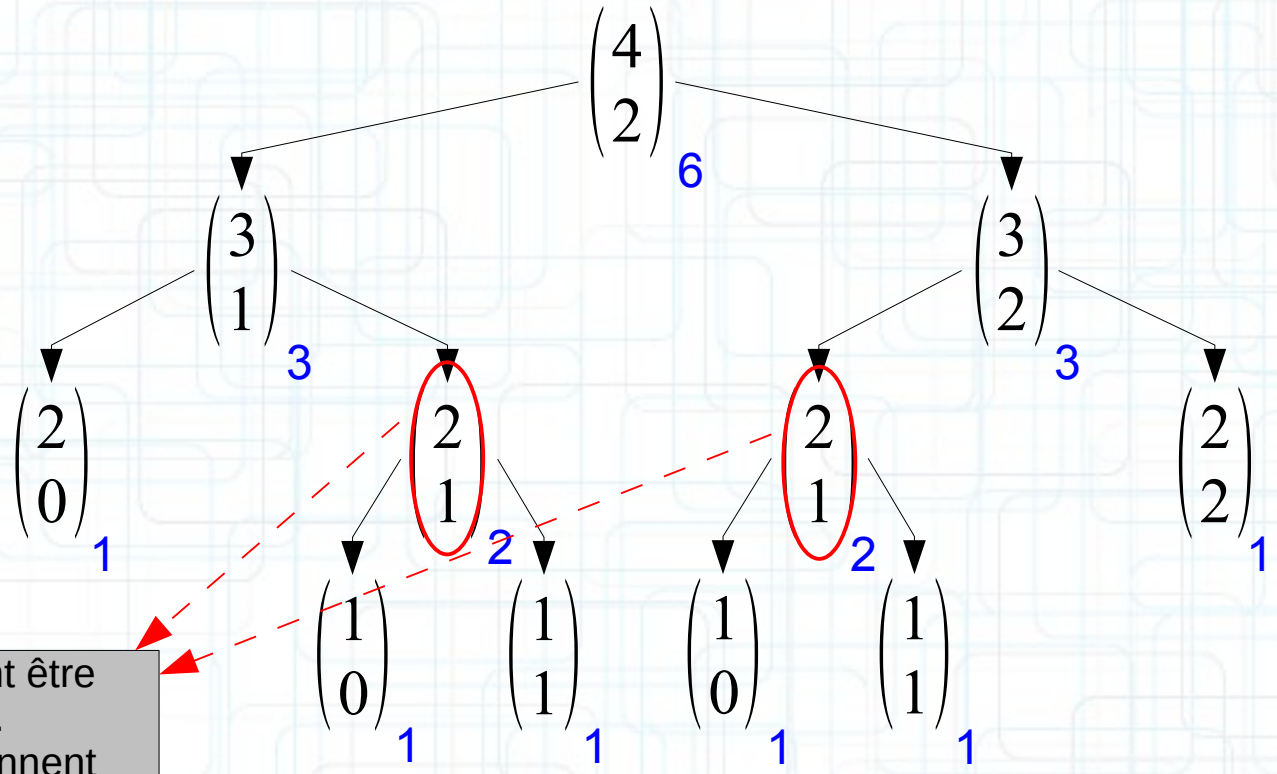


On peut vérifier que le même résultat est obtenu avec la factorielle

$$\binom{4}{3} = \frac{4!}{3!(4-3)!} = \frac{24}{6 \times 1} = 4$$

Tableaux

- **Combinaisons : récurrence**
  - Exemple : calcul du nombre de façons de prendre k=2 éléments parmi n=4.

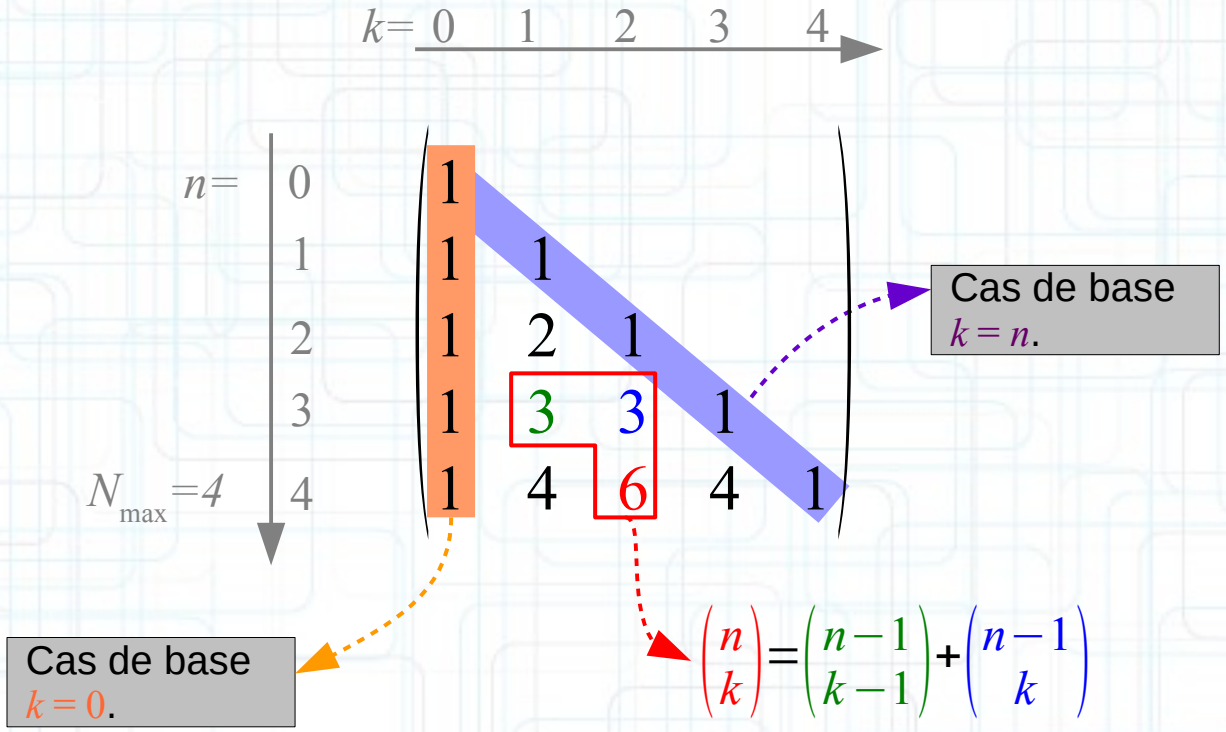


Certains cas peuvent être traités plusieurs fois. Lorsque k et n deviennent grands, le nombre de calculs dupliqués est important !

$$\binom{4}{2}=\frac{4!}{2!(4-2)!}=\frac{24}{2\times 2}=6$$

Tableaux

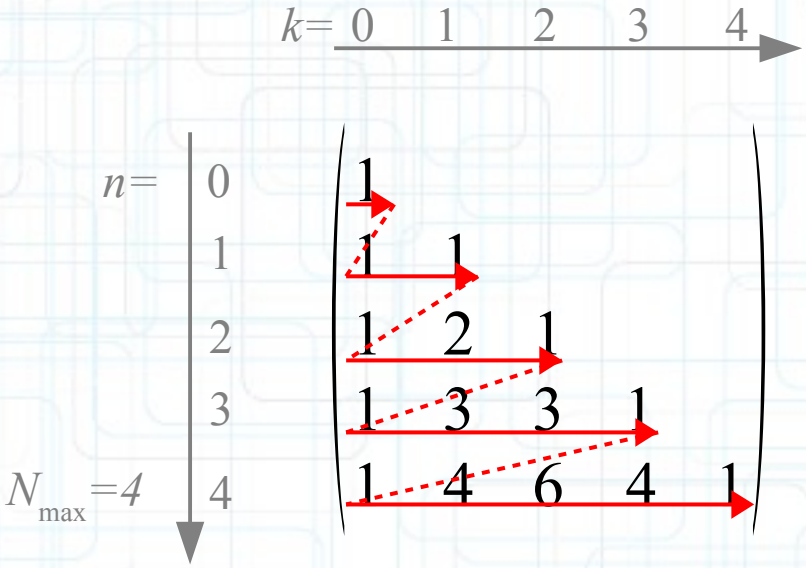
- **Combinaisons : mémoïsation**
  - On peut calculer de façon plus efficace le nombre de combinaisons pour tous les couples (n,k) tels que $0 \leq k \leq n \leq N_{max}$, en calculant la matrice triangulaire suivante(1).



(1) appelée « triangle de Pascal »

Tableaux

- **Combinaisons : mémoïsation**
  - On peut calculer de façon plus efficace le nombre de combinaisons pour tous les couples (n,k) tels que $0 \leq k \leq n \leq N_{max}$, en calculant la matrice triangulaire suivante.



Tableaux

- **Combinaisons : mémoïsation**
  - Construction de la matrice triangulaire des combinaisons, pour $0 \leq k \leq n \leq N_{\max}$

```
final int N_MAX = 7;
int[][] comb= new int[N_MAX+1][];
for (int n= 0; n < comb.length; n++) {
  comb[n]= new int[n+1];
  for (int k= 0; k <= n; k++)
  {
    if ((k == n) || (k == 0))
      comb[n][k]= 1; /* cas de base */
    else
      comb[n][k]= comb[n-1][k-1] +
        comb[n-1][k]; /* cas récursif */
  }
}
```

index extérieur pour n

index intérieur pour k

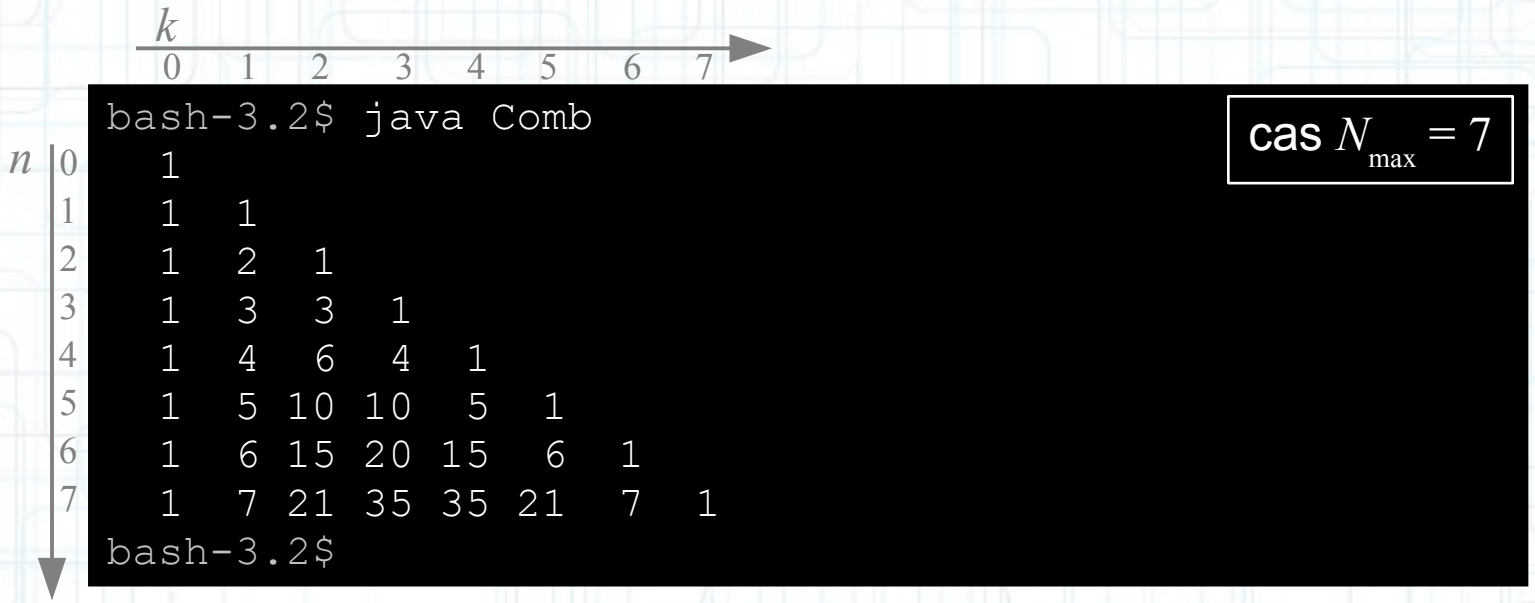
$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Tableaux

- **Combinaisons : mémoïsation**
  - Affichage du tableau résultant.

```
for (int i= 0; i < comb.length; i++) {
  for (int j= 0; j < comb[i].length; j++)
    System.out.printf("%2d", comb[i][j]);
  System.out.println();
}
```



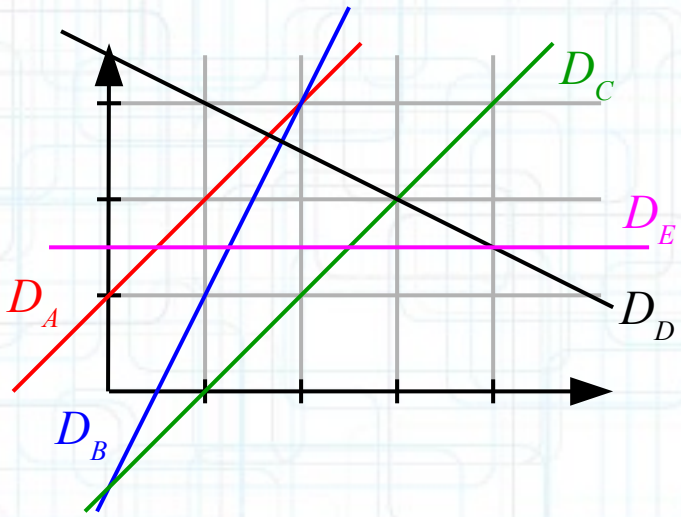
Annexe
– Proposition d'exercices –

Tableaux

- **Exercices**
  - Ecrire une méthode qui identifie les dupliqués dans un tableau d'entiers. Il n'est pas permis de changer l'ordre des éléments du tableau.
  - Ecrire une méthode qui permet d'ajouter une chaîne de caractères dans un tableau de chaînes de caractères. Les chaînes du tableau sont triées selon un ordre lexicographique. La méthode doit maintenir le tableau trié.
  - Ecrire une méthode qui calcule le produit scalaire de deux vecteurs.
  - Ecrire une méthode qui calcule le produit de deux matrices.
  - Ecrire une méthode qui effectue la copie d'un tableau en profondeur pour un nombre quelconque N de dimensions ?
  - Ecrire une méthode qui recherche l'occurrence du contenu d'un tableau dans un autre tableau.

Tableaux multi-dimensionnels

- **Type, déclaration, construction**
  - Exemple : modélisation d'un système d'équations linéaires



$D_A \equiv x - y = -1$

$D_B \equiv 2x - y = 1$

$D_C \equiv x - y = 1$

$D_D \equiv x + 2y = 7$

$D_E \equiv y = \frac{3}{2}$

  - Trouver l'intersection de D_A et D_B
  - Résoudre le système

$$\begin{cases} x - y = -1 & (D_A) \\ 2x - y = 1 & (D_B) \end{cases}$$

Tableaux multi-dimensionnels

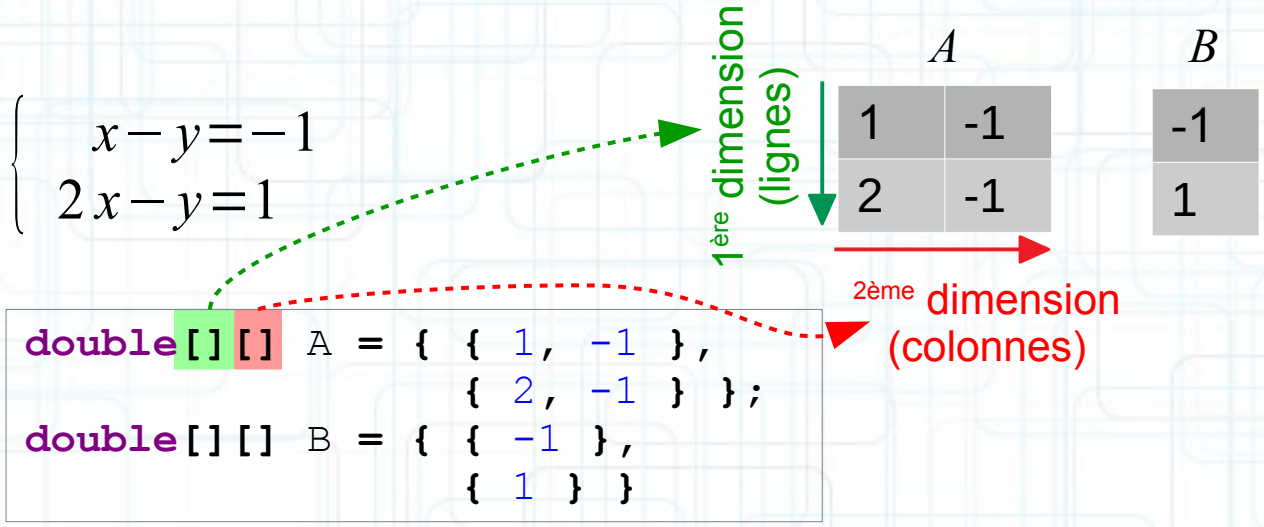
- **Exemple : Système d'équations 2x2**
  - sous forme matricielle

$$Ax = b$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$\begin{cases} a_{1,1}x + a_{1,2}y = b_{1,1} \\ a_{2,1}x + a_{2,2}y = b_{2,1} \end{cases}$$

$$\begin{cases} x - y = -1 \\ 2x - y = 1 \end{cases}$$



Tableaux multi-dimensionnels

- **Exemple : Système d'équations 2x2**
  - Comment afficher le système à la console ?

```
/* Affichage du système d'équations */
final char[] VAR_NAMES = { 'x', 'y' };
for (int i= 0; i < 2; i++) {
  for (int j= 0; j < 2; j++) {
    if (j > 0) {
      char op= (A[i][j] >= 0) ? '+' : '-';
      System.out.printf("%c %.2f", op,
                        Math.abs(A[i][j]));
    } else
      System.out.printf("%.2f", A[i][j]);
    System.out.printf(" * %c", VAR_NAMES[j]);
  }
  System.out.printf(" = %.2f\n", B[i][0]);
}
```

```
1.00 * x - 1.00 * y = -1.00
2.00 * x - 1.00 * y = 1.00
```

Comment améliorer ce programme de façon à supprimer ces *"magic numbers"* ?

Tableaux multi-dimensionnels

- **Exemple : Système d'équations 2x2**
  - résolution par inversion de la matrice A

$$x = A^{-1} b$$

$$A^{-1} = \frac{1}{det(A)} \begin{pmatrix} a_{2,2} & -a_{1,2} \\ -a_{2,1} & a_{1,1} \end{pmatrix}$$

$$det(A) = a_{1,1} a_{2,2} - a_{1,2} a_{2,1}$$

```
double detA =
  A[0][0] * A[1][1]
  - A[0][1] * A[1][0];

double[][] invA =
  { { A[1][1]/detA, -A[0][1]/detA },
    { -A[1][0]/detA, A[0][0]/detA} };
```

Tableaux multi-dimensionnels

- **Exemple : Système d'équations NxN**
  - Comment résoudre le système ?
  - Rappel : méthode d'élimination de Gauss

1) Elimination x dans Eq.2

$$\begin{array}{ll}(\text{Eq.1}) & a_{1,1}x+a_{1,2}y=a_{1,3}\\ (\text{Eq.2}) & a_{2,1}x+a_{2,2}y=a_{2,3}\end{array} \Rightarrow \beta=\frac{a_{2,1}}{a_{1,1}} \Downarrow$$

$$\begin{cases} a_{1,1}x+a_{1,2}y=a_{1,3} \\ a'_{2,2}y=a'_{2,3}\end{cases} \Leftarrow \text{Eq.2}' \leftarrow \text{Eq.2} - \beta\,\text{Eq.1}$$

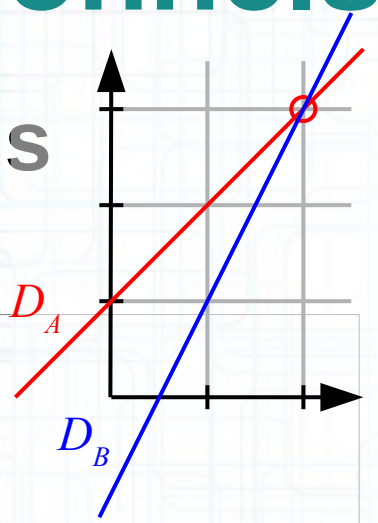
2) Substitution y dans Eq.1

$$\text{Eq.2} \Rightarrow \; y=\frac{a'_{2,3}}{a'_{2,2}} \qquad \text{Eq.1} \Rightarrow \; x=\frac{a_{1,3}-a_{1,2}y}{a_{1,1}}$$

Note : le résumé ne tient pas compte des cas pathologiques (systèmes impossible ou sous-déterminé). De plus, si $a_{1,1}$ vaut zéro, il faut changer de pivot.

Tableaux multi-dimensionnels

- **Exemple : Système d'équations**
  - Comment résoudre le système ?



$$\begin{cases} 1x - y = -1 \\ 2x - y = 1 \end{cases}$$

$$\beta = \frac{2}{1} = 2$$

$$2x - 2x - y - 2(-y) = 1 - 2(-1)$$

$$y = 3$$

$$x = -1 + y$$
$$= -1 + 3$$
$$= 2$$

```
/* (1) forward-elimination
   - supprimer variable 'x' de Eq.1
*/
double beta= sysEq[1][0] / sysEq[0][0];
for (int j= 0; j < NUM_VARS + 1; j++)
  sysEq[1][j]= sysEq[1][j] - beta * sysEq[0][j];

/* (2) back-substitution
   - obtenir 'y' de Eq.2
   - remplacer 'y' dans Eq.1 et obtenir 'x'
*/
double y= sysEq[1][2] / sysEq[1][1];
double x= (sysEq[0][2] - sysEq[0][1] * y) / sysEq[0][0];
```

```
1.00 * x - 1.00 * y = -1.00
2.00 * x - 1.00 * y = 1.00
x = 2.00
y = 3.00
```

Note : le programme proposé ne gère pas les systèmes impossibles ou sous-contraints. Il ne gère pas non plus le cas où un autre pivot doit être trouvé.