Sous-programmes et tableaux

Amélie Lambert

USSE09 - Java

Les sous-programmes

Notion de sous-programme

- **Problème :** Lorsqu'un programme est long on ne peut le coder en entier dans la fonction principale.
- **Idée :** Décomposer le programme en plusieurs sous-programmes plus petits.
- Les sous-programmes (procédures ou fonctions) permettent de découper un gros programme en morceaux plus petits.

Notion de sous-programme

- Un sous-programme possède un nom, des variables, des instructions, un début et une fin.
- l'exécution d'un sous-programme est invoquée dans la fonction principale.
- **Définition :** Un sous-programme est un élément de programme nommé et éventuellement paramétré, que l'on définit afin de pouvoir ensuite l'appeler par son nom en affectant, s'il y a lieu, des valeurs aux paramètres.

Les sous-programmes

- **Les intérêts :**
  - le gain de place en mémoire pour le code du programme : lorsqu'un sous-programme est appelé plusieurs fois, dans une boucle,
  - les notions d'abstraction et de modularité : utiliser un appel de sous-programme permet de d'écrire une application en faisant abstraction (en dissimulant) les détails de la fonction que réalise ce sous-programme.

- **Quelques règles :**
  - Un sous-programme doit être homogène : il doit réaliser une tâche précise, formant un tout.
  - Il doit être de taille "raisonnable". La compréhension et la gestion du programme en dépend.

Les sous-programmes

- La définition du sous-programme est composée d'une spécification qui indique son nom, ses paramètres (ou arguments) avec leurs caractéristiques (nom, type) et d'un corps comprenant éventuellement des déclarations d'objets locaux au sous-programme et les instructions à exécuter.
- Déclaration d'un sous-programme en java :

```
public static typeRetour nomFonction (typeAr1 arg1,
typeAr2 arg2, ... )
```

Les types de sous-programmes

Types de sous-programmes

Il existe deux types de sous-programmes :

- **Les procédures :** sous-programme nommé, ne renvoyant pas de résultat, et modifiant généralement son environnement.
- **Les fonctions :** sous-programme nommé, renvoyant une valeur.

En programmation orientée objet, et notamment en Java, on utilise le terme de **méthode** pour désigner une procédure ou une fonction définie au sein d'une classe.

Les procédures

Les procédures : sous-programme nommé, ne renvoyant pas de résultat, et modifiant généralement son environnement.

Exemple :

```
public void afficher(int i) {
  System.out.println(i);
}

public static void main(String[] args)
{
  afficher(3);
}
```

L'appel d'une procédure constitue une instruction en lui-même

Les fonctions

Les fonctions : sous-programme nommé, renvoyant une valeur.

Exemple :

```
public int carre(int n) {
  return n*n;
}

public static void main(String[] args)
{
  int n;
  n = carre(2);
  afficher(n);
}
```

Les fonctions

Les fonctions exécutent des instructions et en plus retournent une valeur.

L'appel d'une fonction est remplacé dans le programme appelant par la valeur renvoyée.

L'appel d'une fonction représente une valeur qui doit être utilisée à l'intérieur d'une instruction.

Appel de sous-programme

Appel de sous-programmes

```
public void afficher(int e) {
  System.out.println(e);
}
```

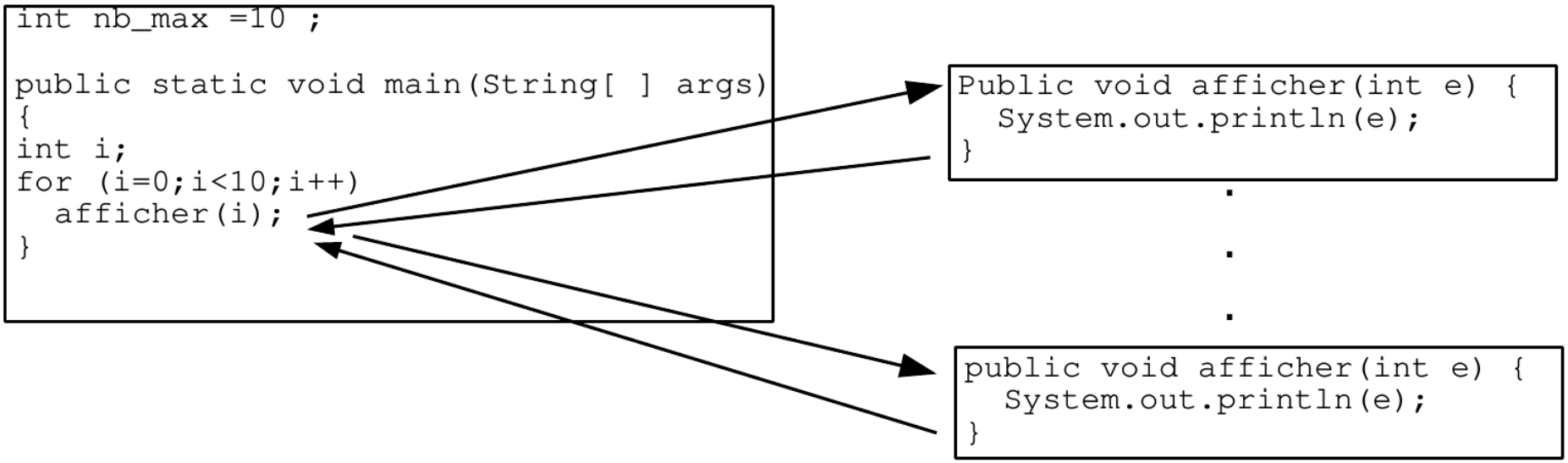
```
int nb_max=10;

public static void main(String[ ] args)
{
  int i;
  for (i=0;i<nb_max;i++)
    afficher(i);
}
```

La procédure `afficher` affiche l'entier e qui lui est passé en paramètre.

La fonction principale `main` appelle la procédure `afficher` 10 fois, pour les valeurs successives 0, 1, 2, ..., 9.

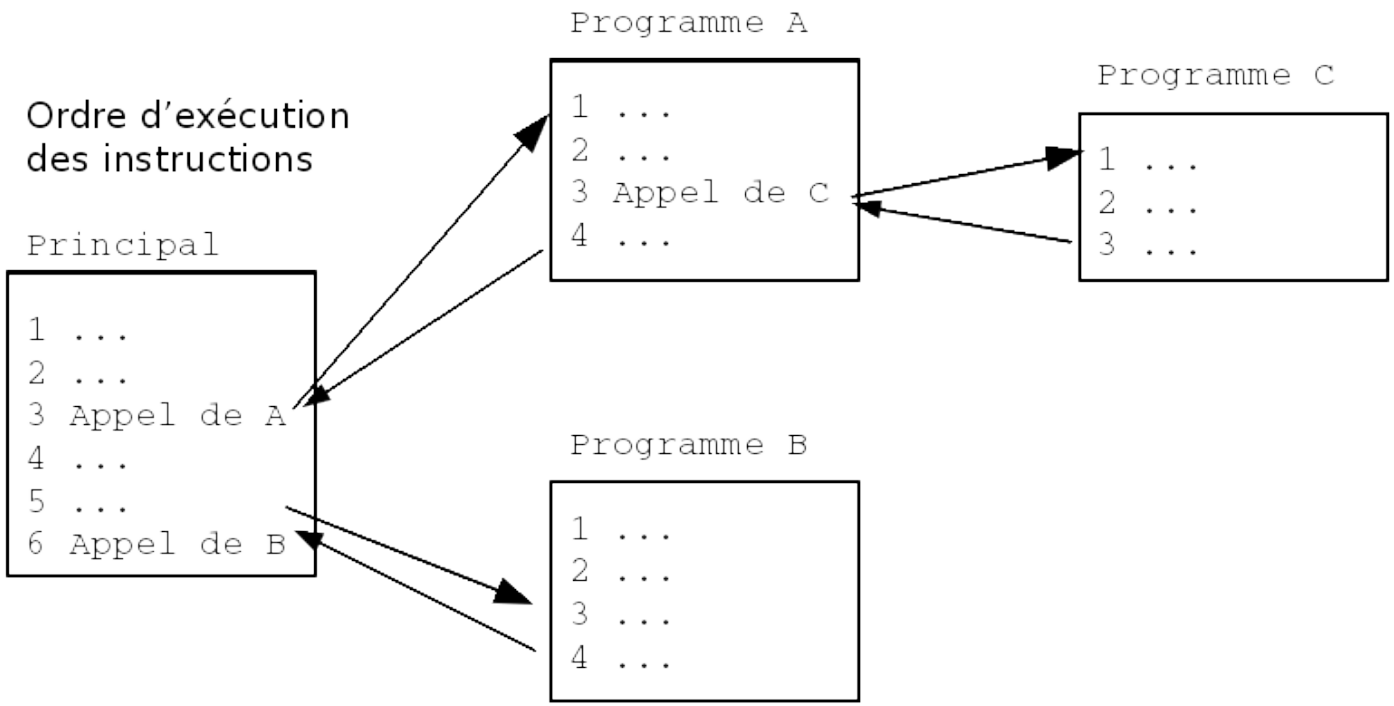
Appel sous-programmes

La procédure `afficher` est appelée 10 fois dans la boucle du programme principal :



Appel sous-programmes en cascade

Un sous-programme peut lui-même appeler un autre sous programme.
A chaque fin de sous-programme, on revient à l'exécution de l'instruction suivante dans le programme appelant.



Communication entre programme appellant et sous-programmes

Les données et les résultats d'un sous-programme proviennent d'une communication entre le programme appelant et le sous-programme.

3 modes de communication sont possibles :

- l'utilisation de variables communes à l'appelant et à l'appelé : ce sont les **variables globales**.
- **le passage de paramètres**, pour les procédures et les fonctions
- **la valeur de retour** pour les fonctions

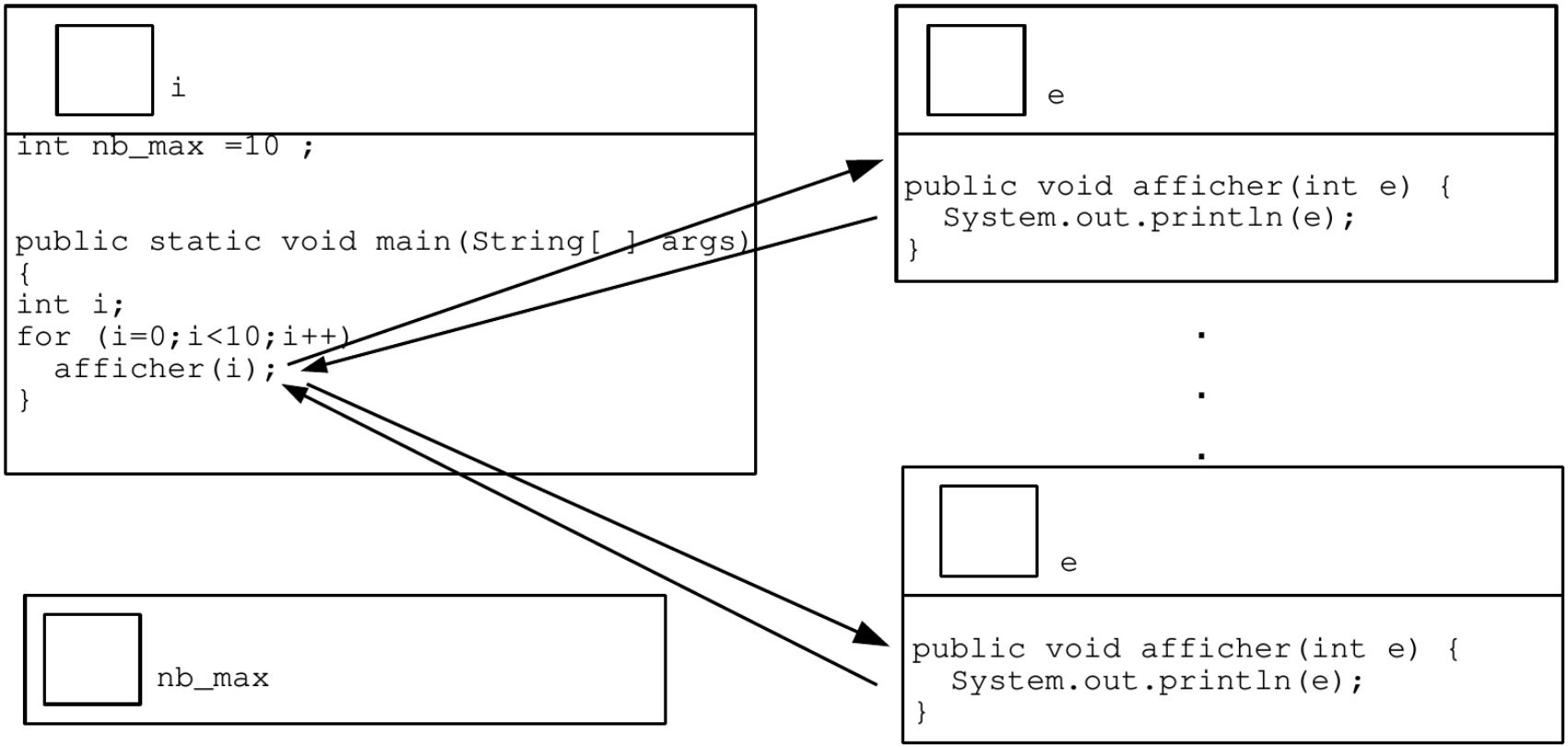
Les variables locales ou globales

L'endroit où est déclaré une variable est très important car il détermine dans quel(s) sous- programme(s) elle va pouvoir être utilisée.

- **Une variable locale** est déclaré dans un sous-programme et n'est utilisable que dans le sous-programme où elle a été déclarée.
- **Une variable globale** est déclarée à l'extérieur du programme principal et des sous-programmes : elle est commune à l'ensemble des sous-programmes et du programme principal et est utilisable partout.

Les variables locales ou globales

Chaque procédure qui est appellée possède son propre espace d'adressage :



Passage de paramètre

Appel d'un sous-programme et passage de paramètre

L'appel d'un sous-programme se fait en mentionnant son nom, suivi des paramètres effectifs figurant entre parenthèses et séparés par des virgules.

Il existe deux mécanismes importants de substitution entre paramètres effectifs et paramètres formels :

- le passage **par valeurs,**
- le passage **par référence.**

Le passage de paramètre par valeur

La valeur du paramètre effectif est recopiée dans le paramètre formel à l'entrée du sous-programme. Le sous-programme travaille sur une copie du paramètre effectif. Celui-ci n'est pas modifié à l'issue du sous-programme.

Le passage de paramètre par valeur : Exemple

```
public static void echanger(int x, int y)
{
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

La procédure `echanger` échange l'entier `x` et l'entier `y` qui lui sont passé en paramètre.

```
public static void main(String[ ] args)
{
int a=3;
int b=5;
System.out.println("avant a="+a+",b="+b);
echanger(a,b);
System.out.println("après a="+a+",b="+b);
}
```

La fonction principale `main` appelle la procédure `echanger` sur les entiers `x` et `y`.

Le passage de paramètre par valeur : Exemple

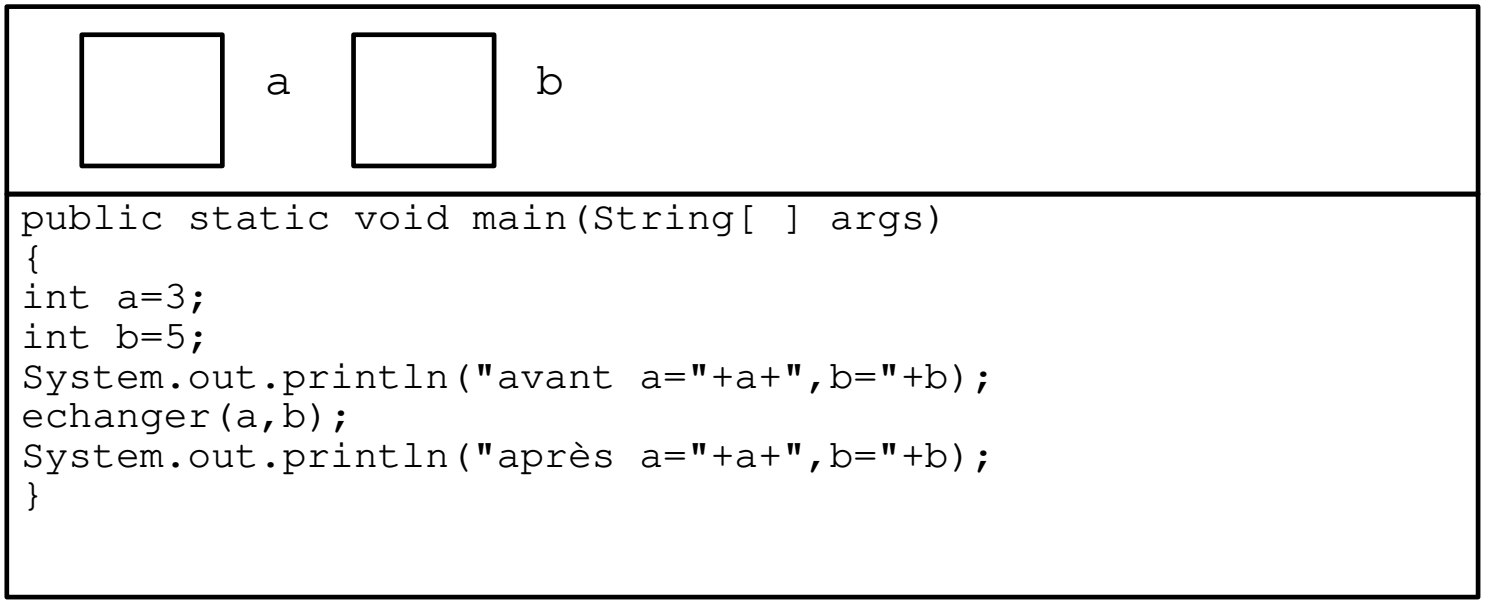
Le résultat de l'exécution du programme est l'affichage suivant :

```
avant échange, a = 3, b= 5
après échange, a = 3, b= 5
```

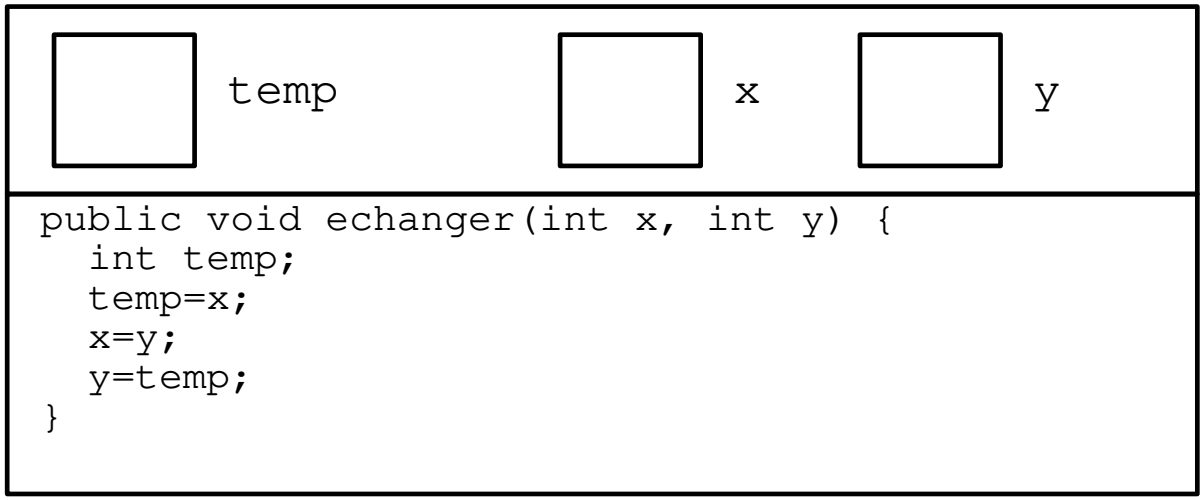
L'appel à la procédure `echanger(a,b)` est un appel par valeur, les paramètres formels `x` et `y` sont remplacés par a=3 et b=5, elle reçoit des copies des paramètres `a` et `b`.

Le passage de paramètre par valeur : Exemple

L'état de la mémoire avant le début de l'exécution du programme :



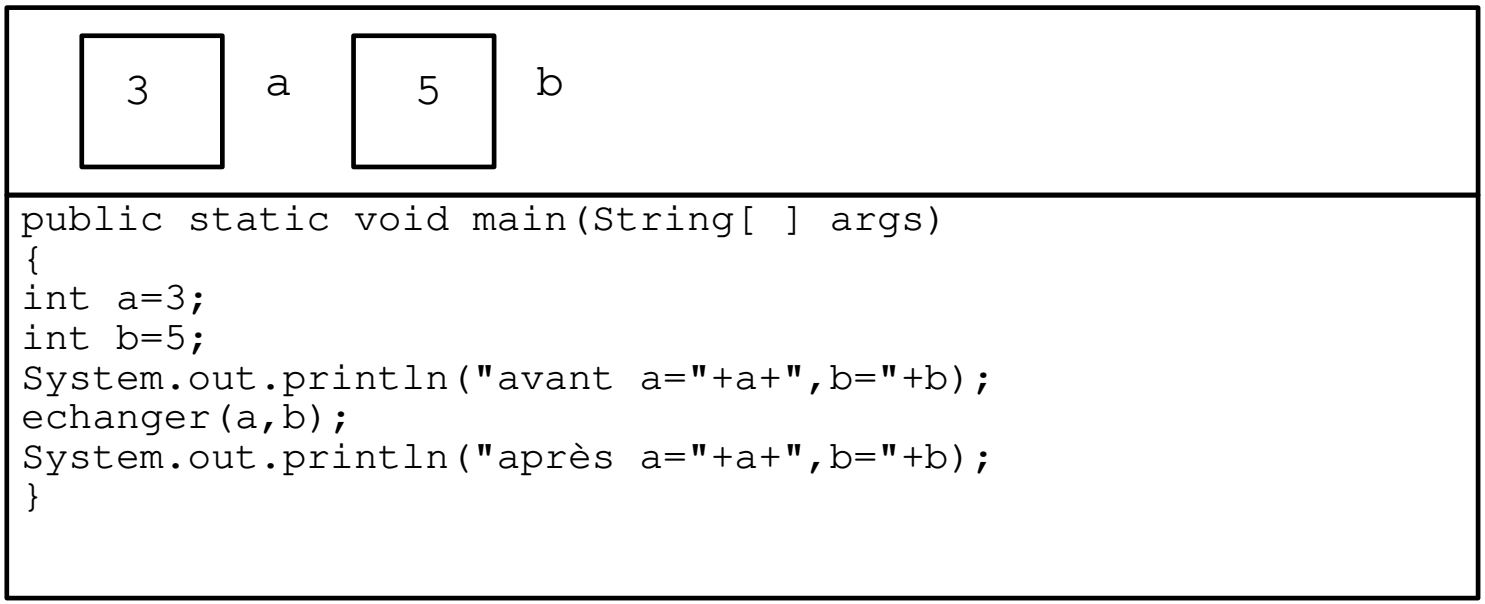

```
public static void main(String[ ] args)
{
int a=3;
int b=5;
System.out.println("avant a="+a+",b="+b);
echanger(a,b);
System.out.println("après a="+a+",b="+b);
}
```



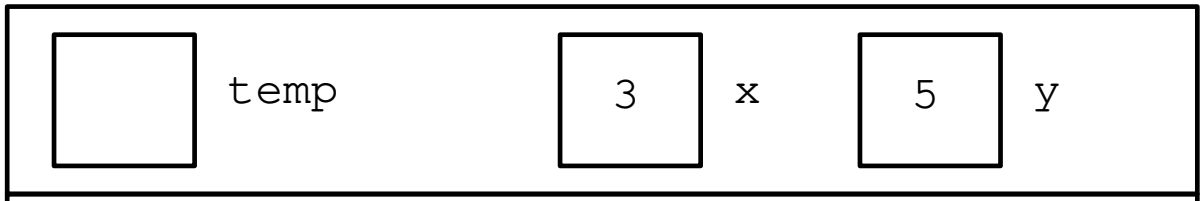

```
public void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par valeur : Exemple

On commence par exécuter les instructions de la procédure principale :



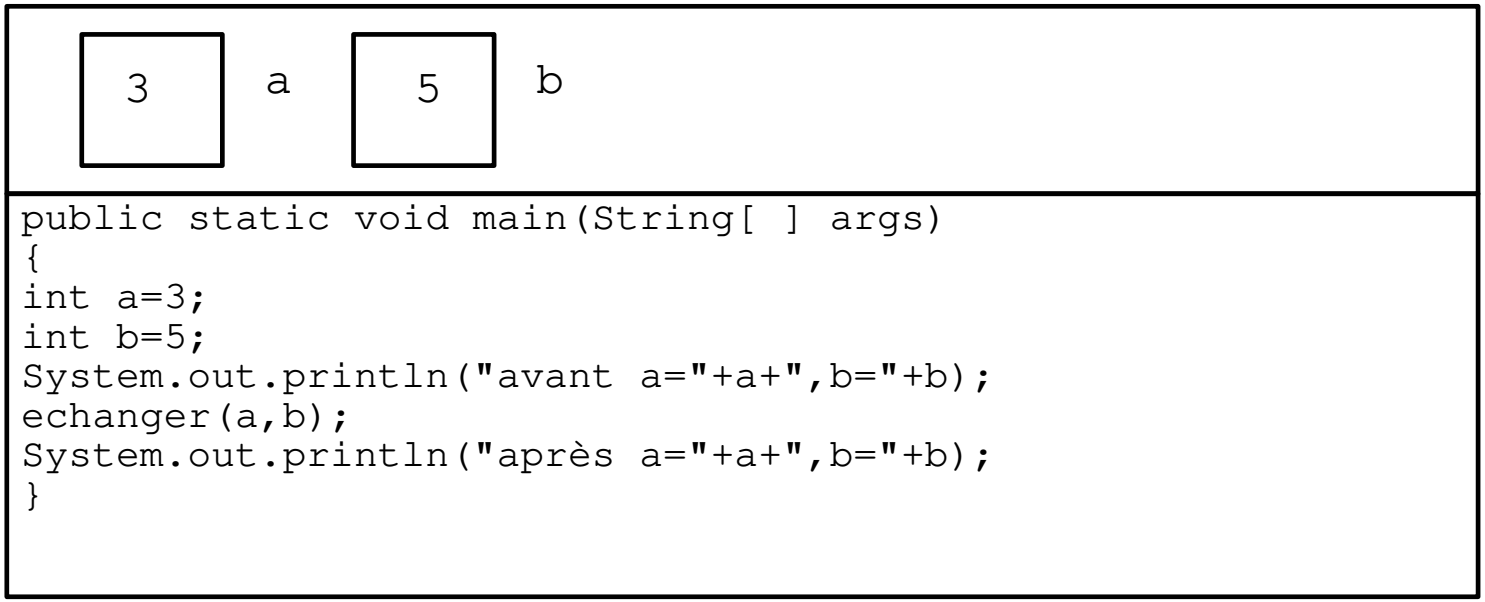

```
public static void main(String[ ] args)
{
int a=3;
int b=5;
System.out.println("avant a="+a+",b="+b);
echanger(a,b);
System.out.println("après a="+a+",b="+b);
}
```



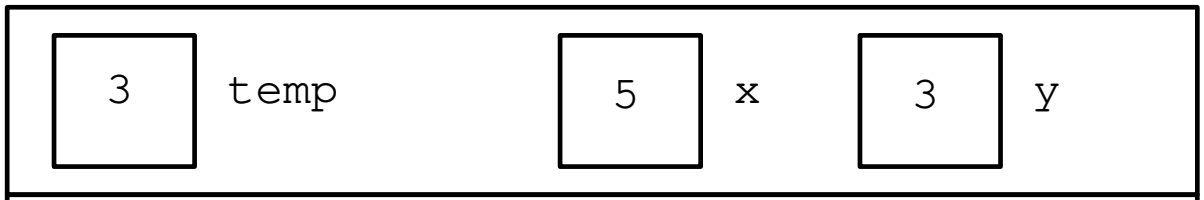

```
public void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par valeur : Exemple

On exécute la procédure `echanger` :



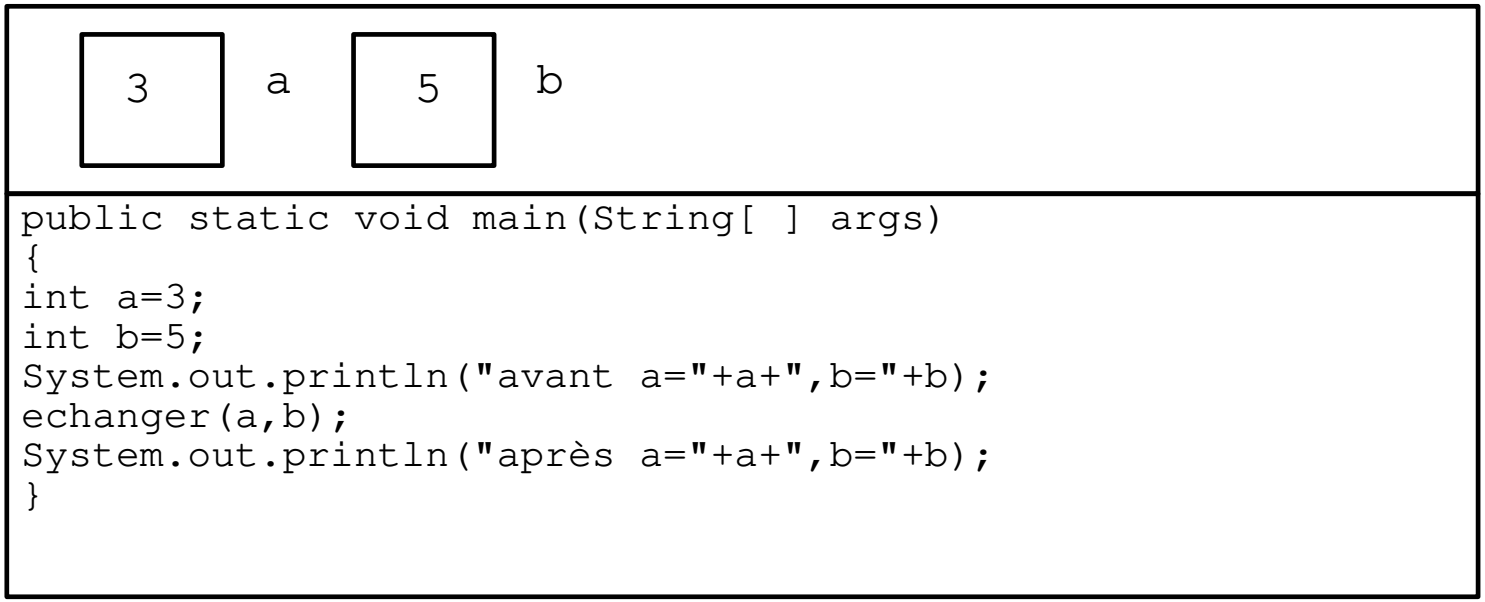

```
public static void main(String[ ] args)
{
int a=3;
int b=5;
System.out.println("avant a="+a+",b="+b);
echanger(a,b);
System.out.println("après a="+a+",b="+b);
}
```



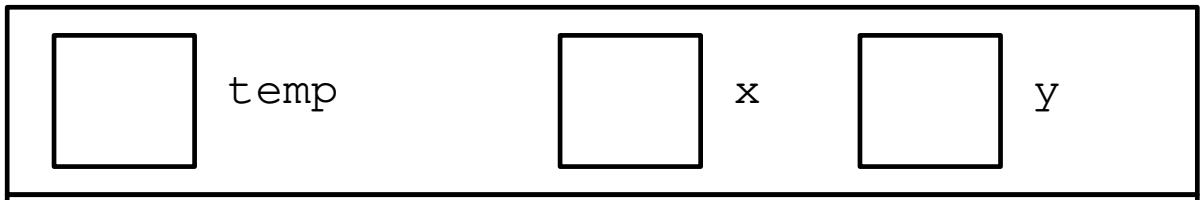

```
public void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par valeur : Exemple

Après l'exécution de la procédure `echanger` :




```
public static void main(String[ ] args)
{
int a=3;
int b=5;
System.out.println("avant a="+a+",b="+b);
echanger(a,b);
System.out.println("après a="+a+",b="+b);
}
```




```
public void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par référence

Définition : une variable de type référence est une variable dans laquelle est stockée une adresse.

L'adresse du paramètre effectif est communiquée au sous-programme qui travaille alors directement sur la variable passée en paramètre et non sur une copie locale. Le sous-programme peut alors modifier la valeur d'une variable du programme qui l'a appelé.

Lorsqu'un tableau est passé en paramètre, c'est sa référence qui est en fait passée (i.e. son adresse), le tableau d'origine et celui vu par la procédure occupent donc le même espace mémoire.

Le passage de paramètre par référence : Exemple

La procédure `echangerTab` échange les élements des d'indices `i` et `j` du tableau `tab` qui lui sont passé en paramètre.

```
public void echangerTab(int t[], int i, int j) {
  int temp;
  temp = t[i];
  t[i]=t[j];
  t[j]=temp;
}
```

La fonction principale `main` appelle la procédure `echangerTab` sur le tableau `tab` et les indices 1 et 3.

```
public static void main(String args[]) {
  int tab[]= {3, 4, 24};
  System.out.println("av tab="+tab[0]+","+tab[1]+","+tab[2]);
  echangerTab(tab,1,2);
  System.out.println("ap tab="+tab[0]+","+tab[1]+","+tab[2]);
}
```

Le passage de paramètre par reférence : Exemple

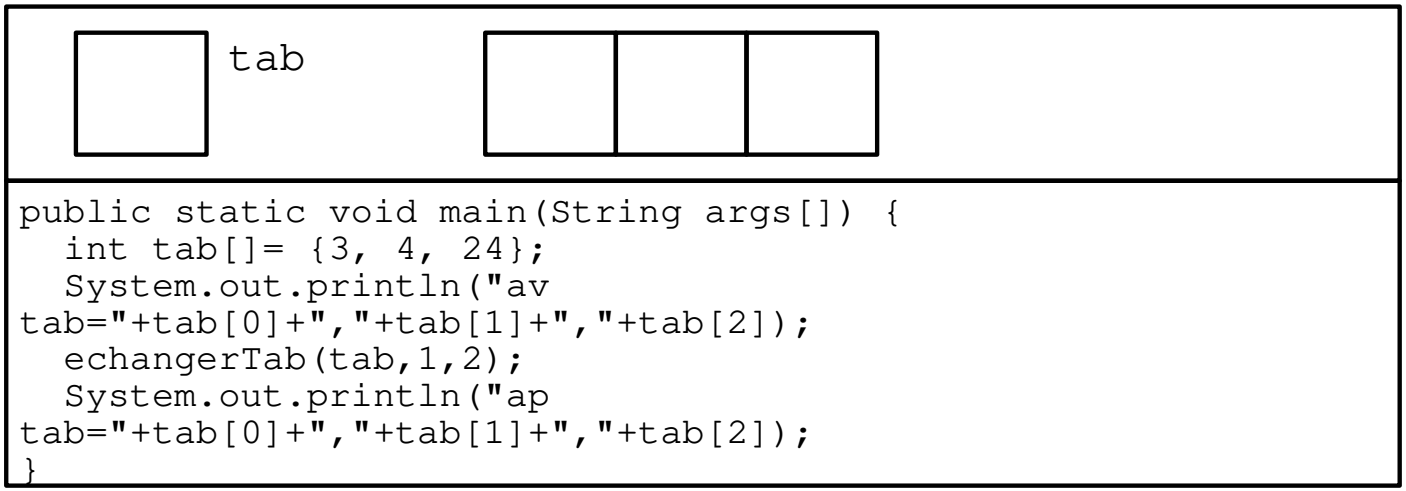
Le résultat de l'exécution du programme est l'affichage suivant :

```
av t =3,4,24
ap t =3,24,4
```

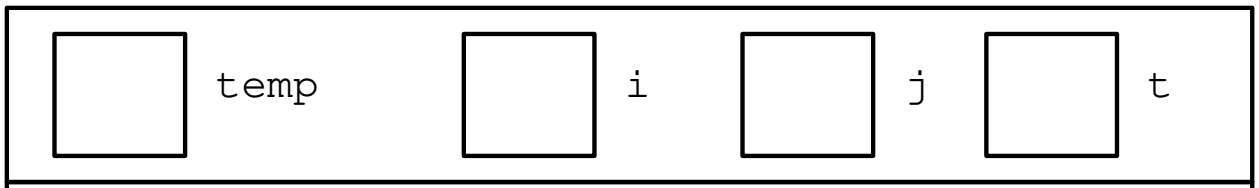
L'appel à la procédure `echangerTab(tab,1,2)` est un appel par valeur, mais la valeur de `tab` est sa référence, les paramètres formels `tab`, `i` et `j` sont remplacés par `ref tab`, `i=1` et `j=4`, elle reçoit des copies de ces paramètres.

Le passage de paramètre par référence : Exemple

L'état de la mémoire avant le début de l'exécution du programme :



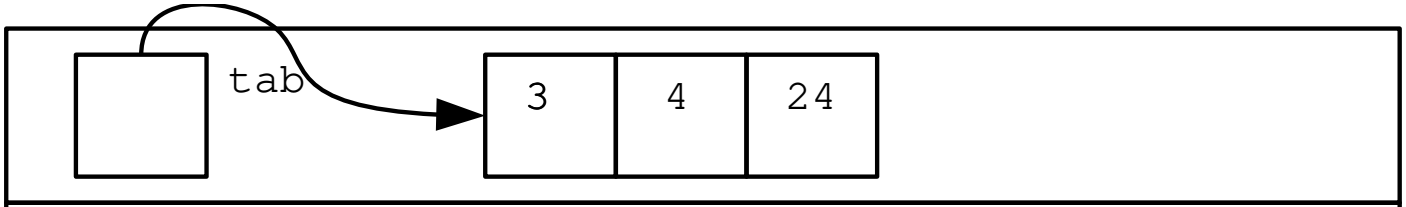

```
public static void main(String args[]) {
  int tab[]= {3, 4, 24};
  System.out.println("av
tab="+tab[0]+","+tab[1]+","+tab[2]);
  echangerTab(tab,1,2);
  System.out.println("ap
tab="+tab[0]+","+tab[1]+","+tab[2]);
}
```



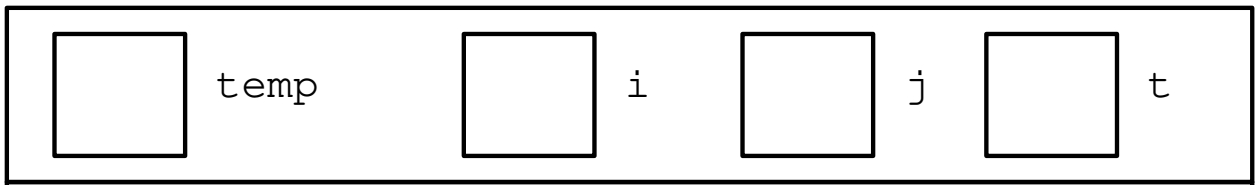

```
public void echangerTab(int t[],int i,int j)
{
  int temp;
  temp = t[i];
  t[i]=t[j];
  t[j]=temp;
}
```

Le passage de paramètre par référence : Exemple

On commence par exécuter les instructions de la procédure principale :



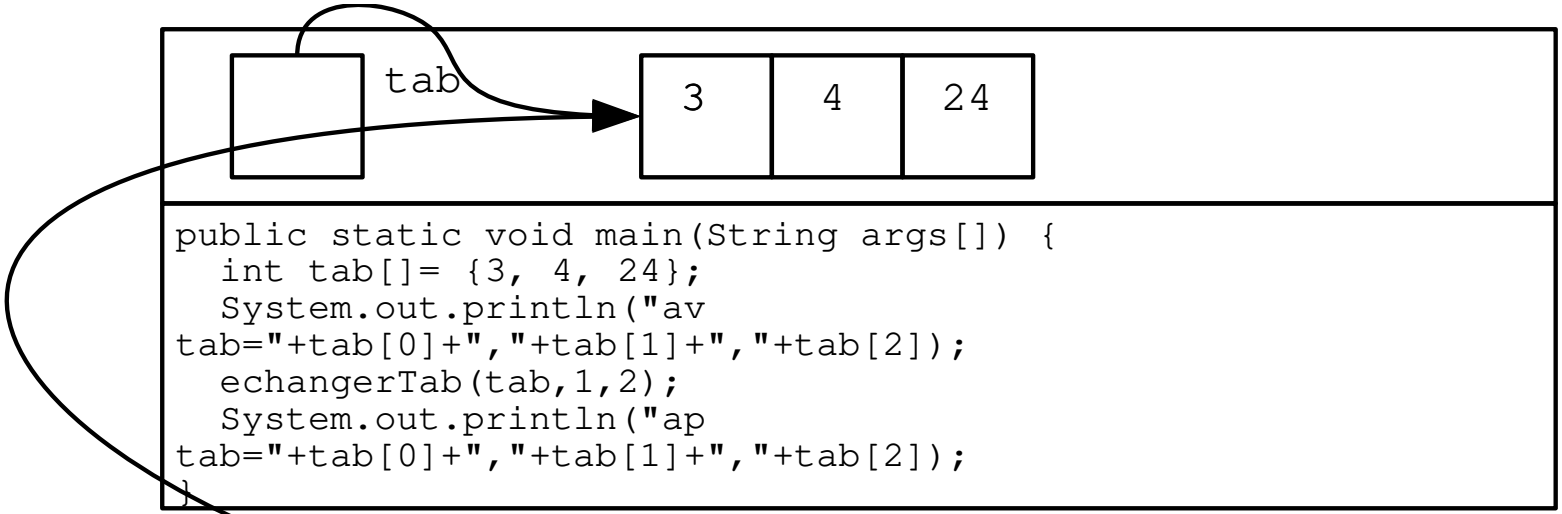

```
public static void main(String args[]) {
  int tab[]= {3, 4, 24};
  System.out.println("av
tab="+tab[0]+","+tab[1]+","+tab[2]);
  echangerTab(tab,1,2);
  System.out.println("ap
tab="+tab[0]+","+tab[1]+","+tab[2]);
}
```

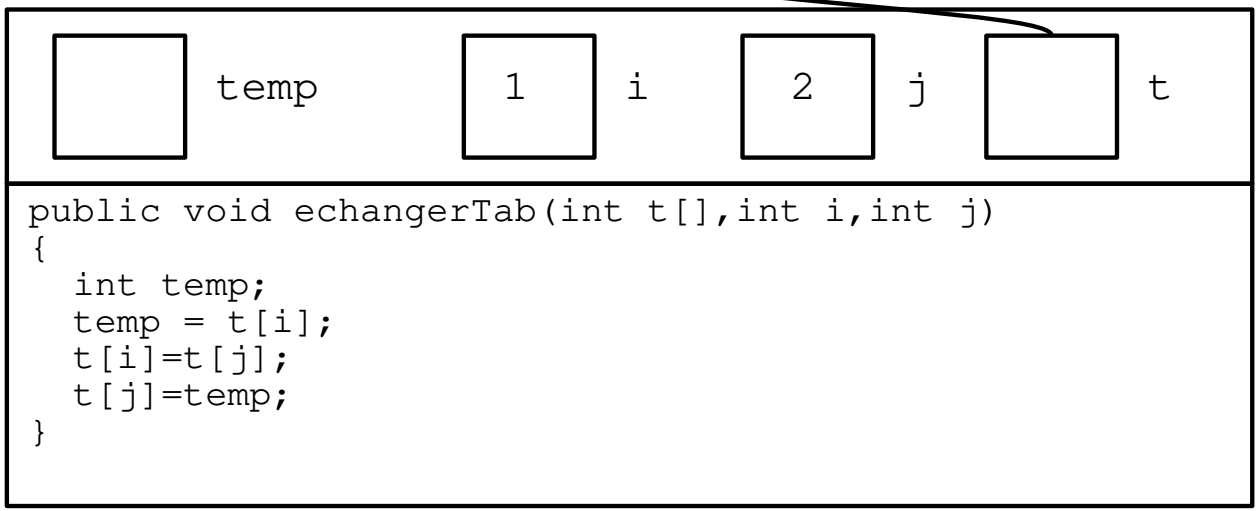



```
public void echangerTab(int t[],int i,int j)
{
  int temp;
  temp = t[i];
  t[i]=t[j];
  t[j]=temp;
}
```

Le passage de paramètre par référence : Exemple

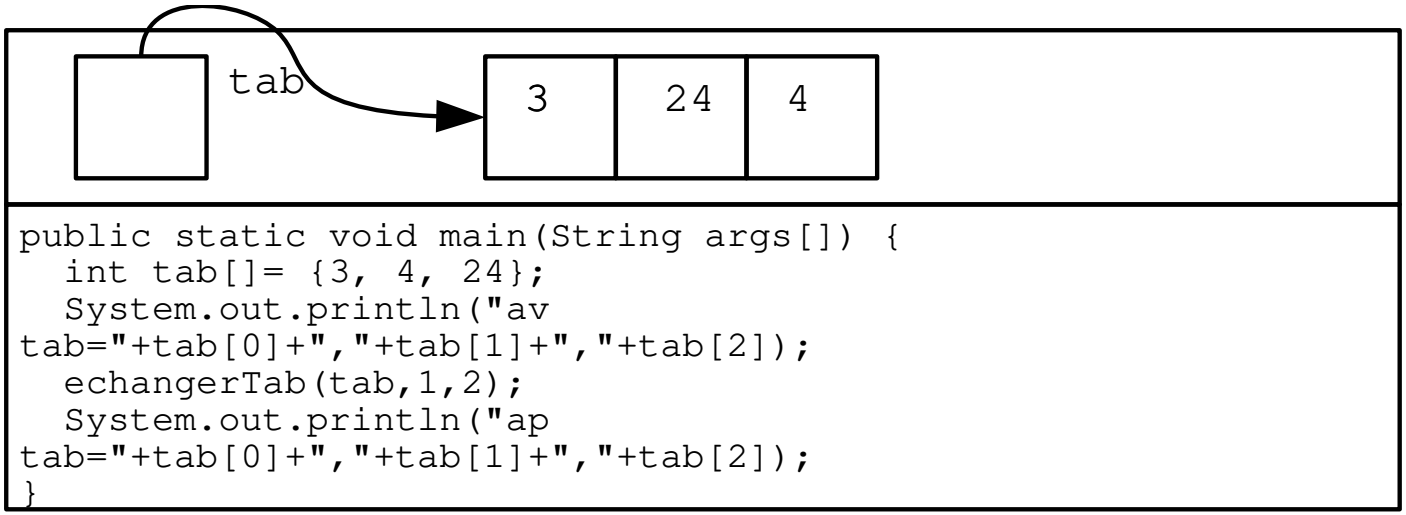
On exécute la procédure `echangerTab` :



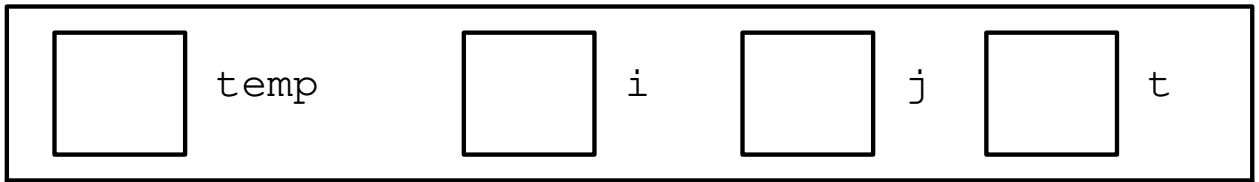


Le passage de paramètre par référence : Exemple

Après l'exécution de la procédure `echangerTab` :




```
public static void main(String args[]) {
  int tab[]= {3, 4, 24};
  System.out.println("av
tab="+tab[0]+","+tab[1]+","+tab[2]);
  echangerTab(tab,1,2);
  System.out.println("ap
tab="+tab[0]+","+tab[1]+","+tab[2]);
}
```



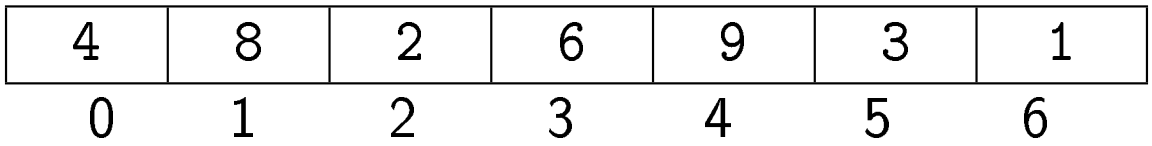

```
public void echangerTab(int t[],int i,int j)
{
  int temp;
  temp = t[i];
  t[i]=t[j];
  t[j]=temp;
}
```

Les tableaux

Notion de tableau

- Pour pouvoir écrire des programmes un peu plus intéressants, il est nécessaire d'appliquer des traitements à des séquences de données. Une manière classique est de les rassembler dans un **tableau**.

- Dans l'exemple qui suit, on réunit dans un tableau le nombre d'heures d'ensoleillement de chaque jour de la semaine.



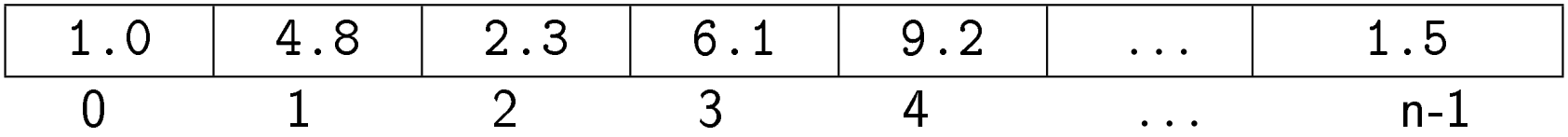
Les **indices du tableau** représentent les jours de la semaine.

Les tableaux en Java (1/2)

- Un **tableau** est donc une **structure de données** qui réunit des valeurs (données) d'un **même type** (le type `int` dans l'exemple).
- On peut le voir comme une suite de cases contiguës repérées (indicées) par un entier (`int`). Le **premier indice** ayant pour valeur `0`.
- Un tableau constitue une nouvelle valeur. Or toute valeur doit appartenir à un type. Il est donc nécessaire de définir un nouveau type auquel ces éléments (ces valeurs) appartiendront.
- Dans l'exemple, on définit le **type** `int[]` comme un ensemble de tableaux contenant des entiers (du type `int`). Les indices de 0 à 6 correspondront respectivement aux jours de la semaine de lundi à dimanche.
- Il est alors possible de déclarer une variable (`uneSemaine`) de ce nouveau type, d'enregistrer des valeurs dans les cases de ce tableau, de sélectionner une case connaissant son indice (`uneSemaine[i]`).

Les tableaux en Java (2/2)

```
float[] t = new float[n]
```



Le premier indice est 0 et le dernier est `t.length -1`.

```
// si t[3] = 1.5f
t[3] = t[3] + 2.5f
// alors t[3] = 4.f
```

Un indice en dehors des bornes du tableau ($0 \ldots \texttt{length} - 1$) provoque une erreur à l'exécution. L' exception : `ArrayIndexOutOfBoundsException` est levée

Création de tableaux en Java (1/2)

- **Déclaration** (allocation du mot mémoire)

```
boolean[] t;
```

- **Création** (allocation en mémoire de 5 composantes de type booléen (1 octet * 5)

```
new boolean[5];
```

- **Déclaration avec initialisation**

```
boolean[] t = {true,false,false};
```

ou

```
boolean[] t = new boolean[3];
t[0]=true;t[1]=false;t[2]=false;
```

Création de tableaux en Java (2/2)

- **ATTENTION**

```
int[] source;
source = {6,78,9,0}; // impossible
```

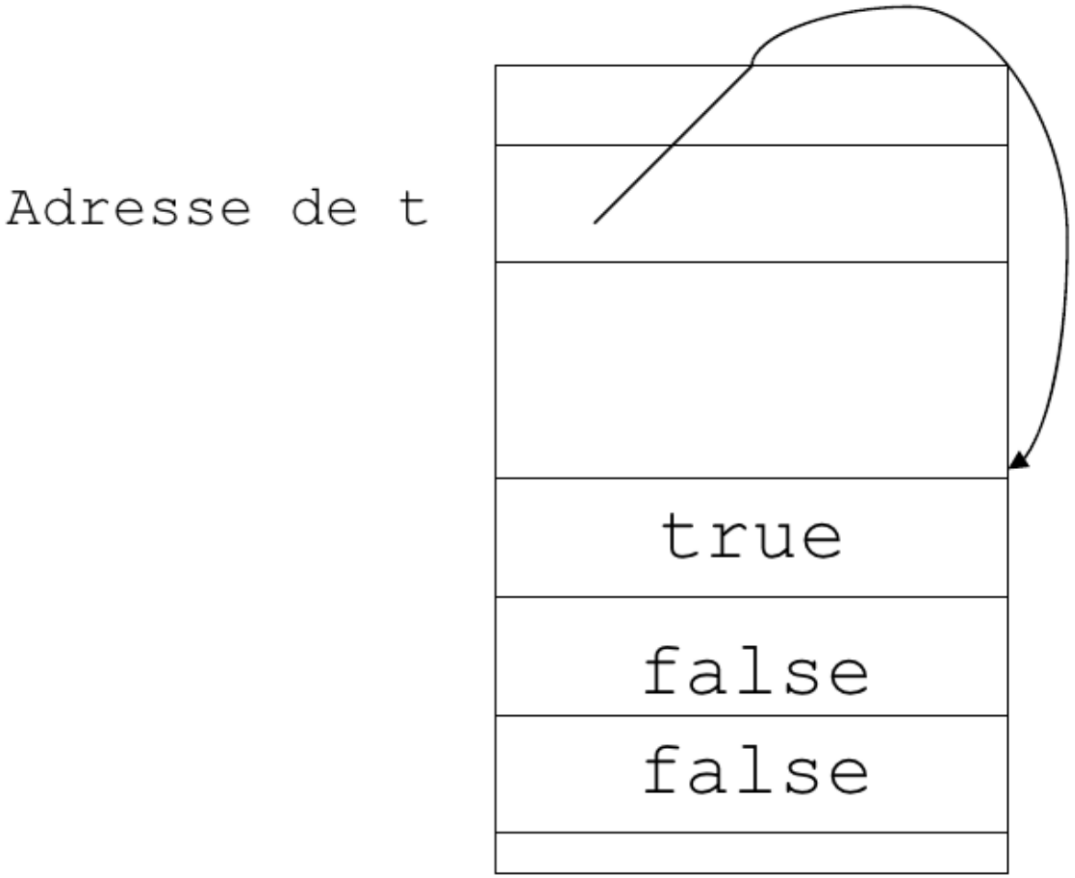
- **MAIS** constructions possibles :

```
int[] source;
source = new int[]{6,78,9,0};
```

- et ensuite

```
source = new int[]{12,42,9,76,0,0,65};
```

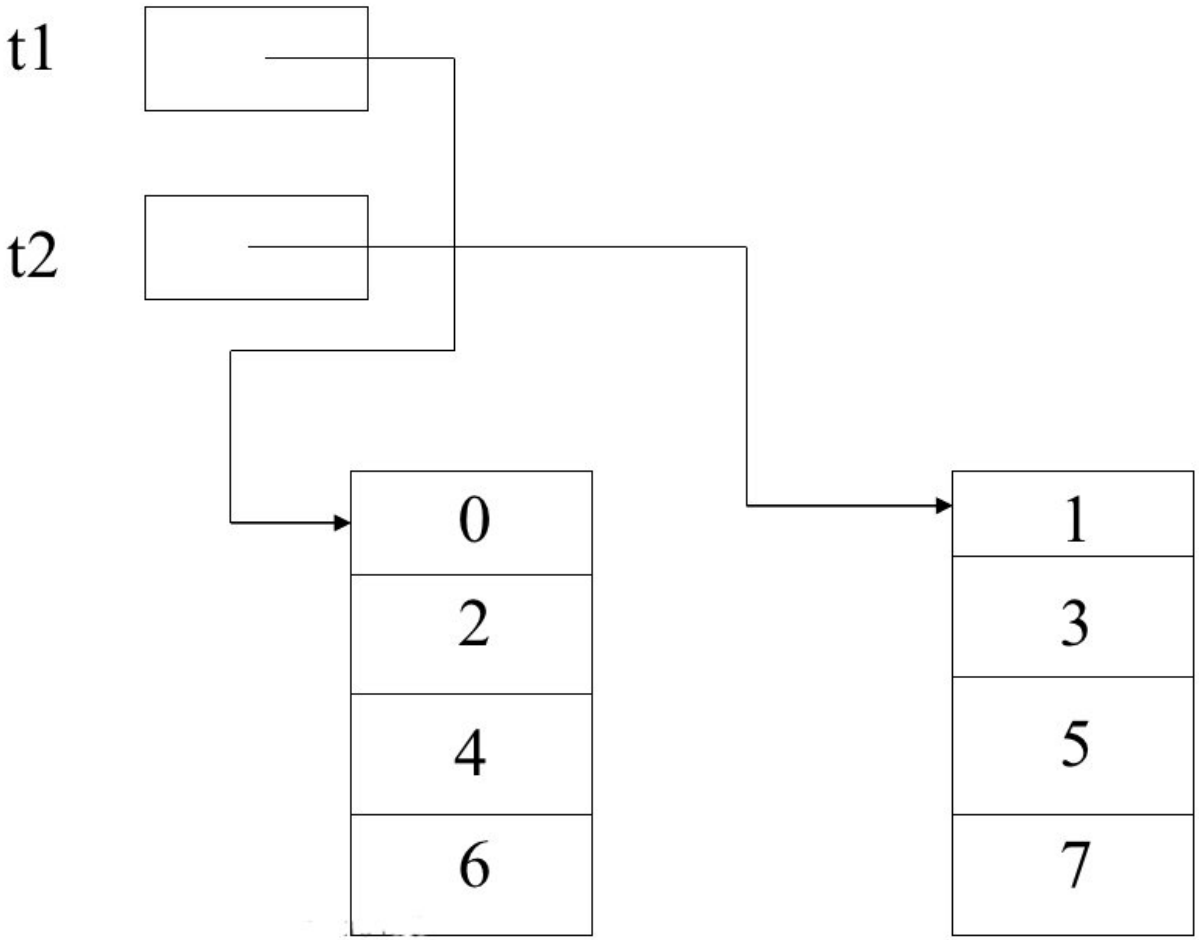
Allocation memoire d'un tableau (1/2)



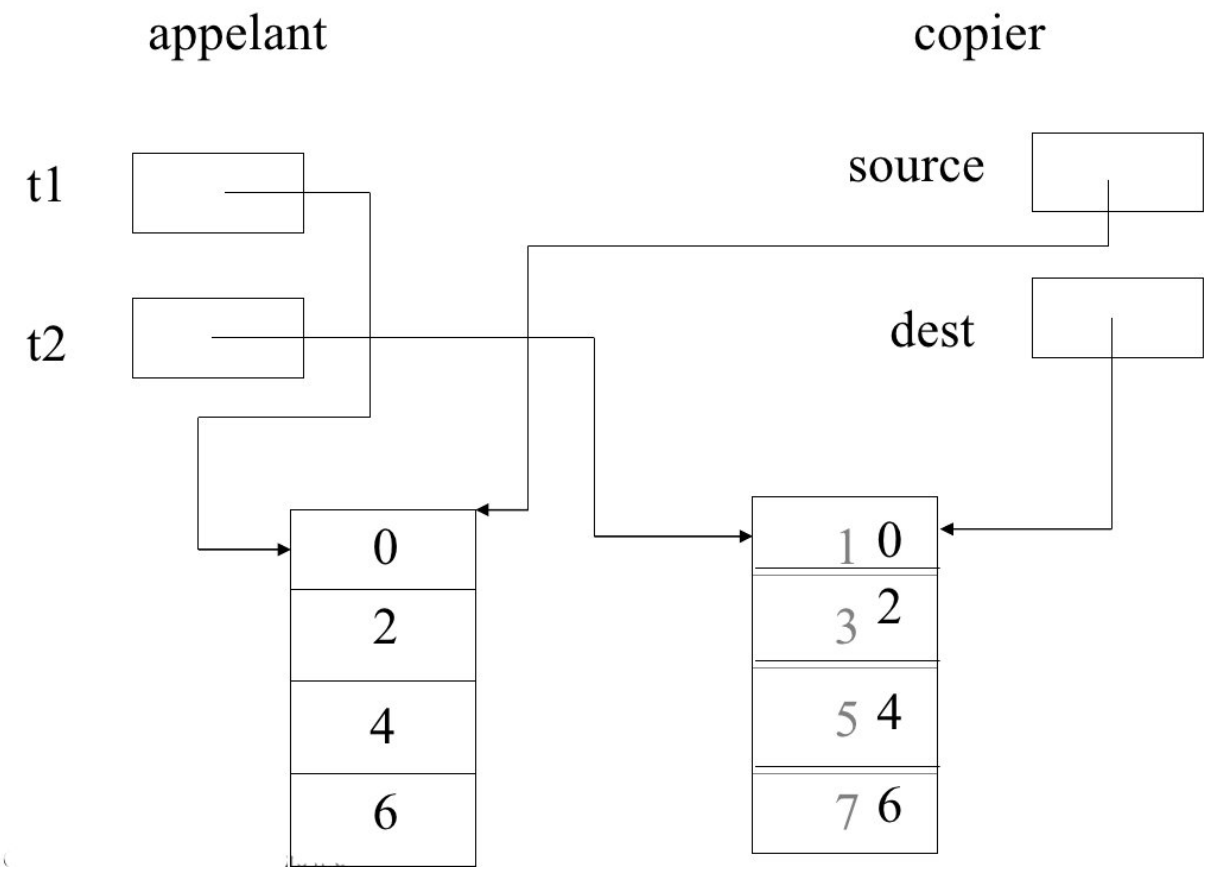
Utilisation d'un tableau, appel de fonction

```
void copier( int[] source, int[] dest ){
    for(int i = 0;i<source.length;i++)
        dest[i] = source[i];
}

void appel(){
    int[] t1 = {0,2,4,6};
    int[] t2 = {1,3,5,7};
    copier(t1,t2);
}
```

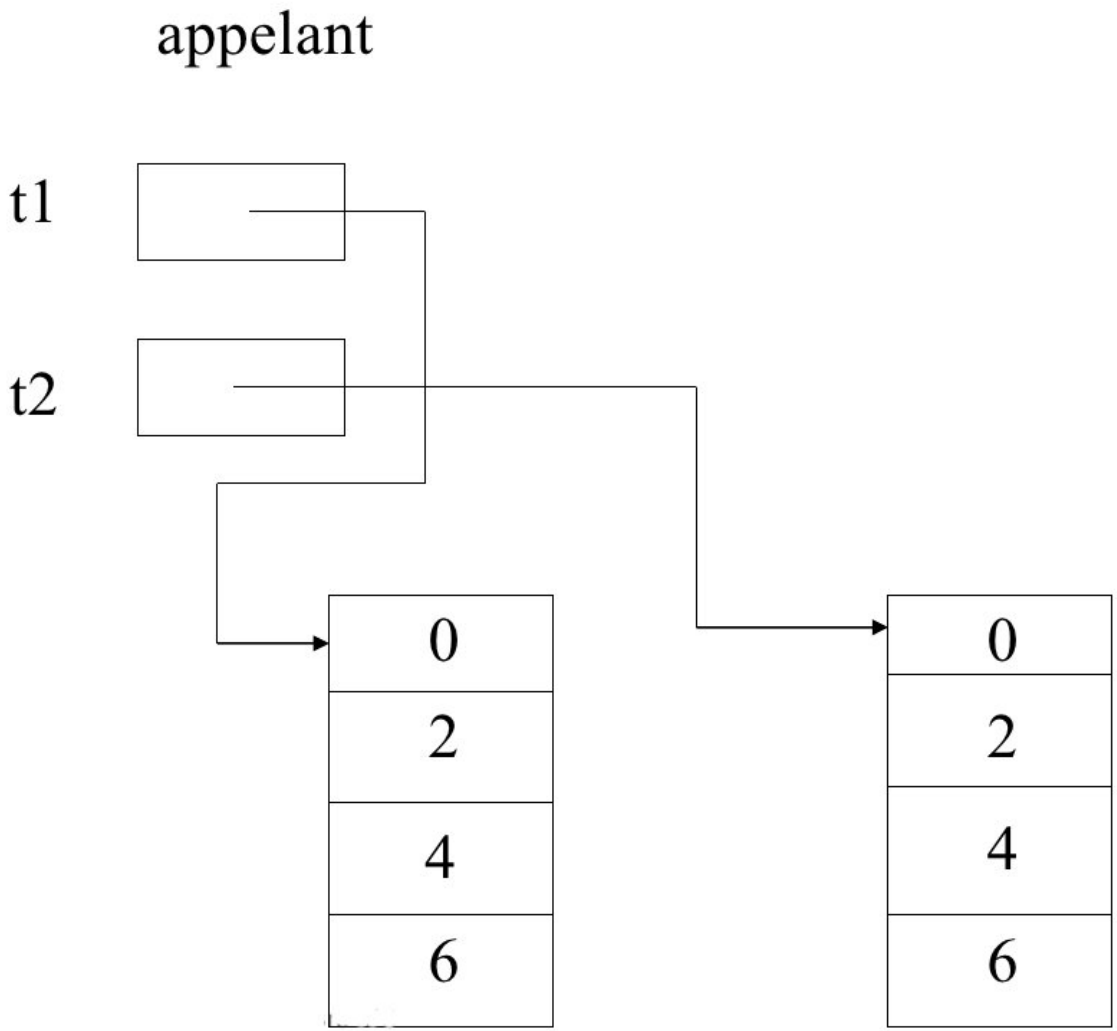
Avant l'exécution de `copier(...)`



En fin d'exécution de `copier(...)`



Retour vers la fonction appel()



Affichage d'un tableau

```
public class AfficherTableau{
  public static void main(String[] args){
    int[] tab = {10,20,30,40};
    for( int i=0;i<tab.length;i++)
      System.out.println("tab["+i+"] "+tab[i]);
  }
}
```

résultat affiché

```
tab[0] = 10
tab[1] = 20
tab[2] = 30
tab[3] = 40
```

résultat affiché si erreur sur la condition de sortie de boucle : `i<=tab.length` :

```
ArrayIndexOutOfBoundsException : 4
```

Un exemple (1/3)

Dans l'exemple qui suit, on implante l'algorithme :

- Saisie dans un tableau du nombre d'heures d'ensoleillement de chaque jour d'une semaine.
- Calcul de la journée la plus ensoleillée
- Affichage à l'écran du résultat

Un exemple (2/3)

```
// nombre d'heures d'ensoleillement journalier d'une semaine
import java.util.Scanner;
public class Soleil{
  final int LUNDI = 0;
  final int MARDI = 1;
  final int MERCREDI = 2;
  final int JEUDI = 3;
  final int VENDREDI = 4;
  final int SAMEDI = 5;
  final int DIMANCHE = 6;

  public static void main(String[] args){
    int[] uneSemaine = new int[7];
    Scanner input = new Scanner( System.in );
    // construction du tableau uneSemaine
    for( int i=0;i<uneSemaine.length;i++){
      System.out.print( "taper le nb d'heures de soleil" + i + " : "
      uneSemaine[i] = input.nextInt();
      System.out.println();
    }
```

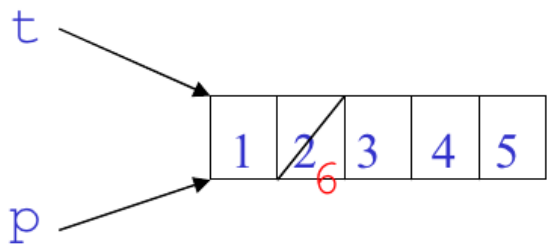
Un exemple (3/3)

```
// calcul du jour le plus ensoleille
  int nbHeures = uneSemaine[LUNDI];
  int unJour   = LUNDI;
  for( int x=MARDI;x<=DIMANCHE;x++){
    if (uneSemaine[x]>nbHeures){
      nbHeures = uneSemaine[x];
      unJour   = x;
   }
  }


// affichage du résultat
  System.out.print( "le numéro du jour le plus ensoleille fut : " );
  System.out.print( unJour + " avec " );
  System.out.print( uneSemaine[unJour] );
  System.out.println( " heures d'ensoleillement" );
 }
}
```

Affectation – Egalité entre tableaux

Affectation entre tableaux

```
int[] t ;
int[] p = {1,2,3,4,5};
t = p; // t et p désignent le même tableau
System.out.println("t[1]="+t[1]);
System.out.println("p[1]="+p[1]);
t[1] = 6;
System.out.println("nouveau t[1]="+t[1]);
System.out.println("nouveau p[1]="+p[1]);
}
```



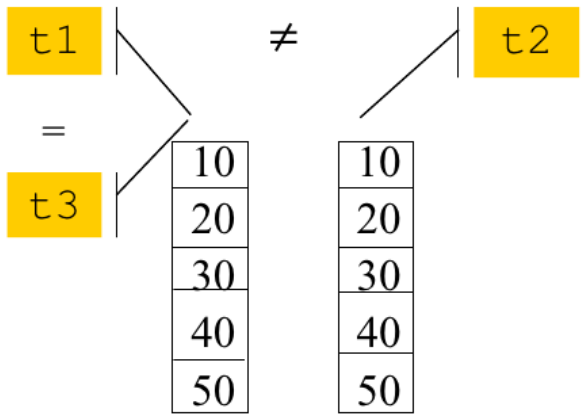
résultat affiché

```
t[1] = 2
p[1] = 2
nouveau t[1] = 6
nouveau p[1] = 6
```

Egalité entre tableaux

La comparaison entre 2 tableaux porte sur leur référence et **non sur leur contenu**.

```
int[] t1 = new int[5]{10,20,30,40,50};
int[] t2 = new int[5]{10,20,30,40,50};
if(t1==t2)
  System.out.println("t1=t2");
else
  System.out.println("t1!=t2");
int[] t3 = t1;
if(t3==t1)
  System.out.println("t3=t1");
else
  System.out.println("t3!=t1");
```



Résultat affiché :

```
t1!=t2
t3=t1
```

Transmission de tableaux en paramètre

```
public class Test{
  public static void main(String[] args){
    int x = 1;
    int[] y = new int[1];
    m(x,y);
  }

  static void m(int n, int[] t){
    n = 150;
    t[0] = 76;
  }
}
```

les paramètres sont passés par valeur, x est une valeur et y est une référence

Tableaux multidimensionnels

Tableaux multidimensionnels

Un tableau multidimensionnel est vu comme un **tableau de tableaux**.

Exemple :

```
double[][] precipitation = new double[13][32];
```

`precipitation` est un tableau de 13 composants, chacun d'entre eux étant un tableau de 32 composants.
Il peut, par exemple, représenter la précipitation de pluie pour chaque jour de chaque mois d'une année.

Le niveau de précipitation du 23 mars sera représenté par :

```
precipitation[3][23]
```

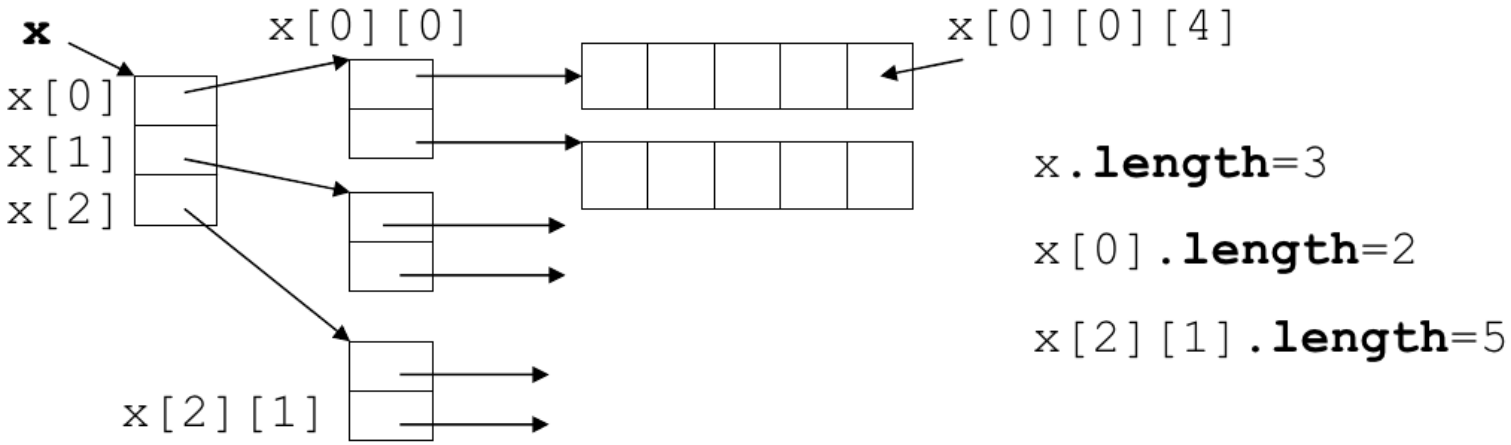
à condition que les lignes et colonnes d'indice 0 soient initialisées à 0

La longueur du tableau représentant le mois d'avril sera connue par :

```
precipitation[4].length
```

Longueur des tableaux multi-dimensionnels

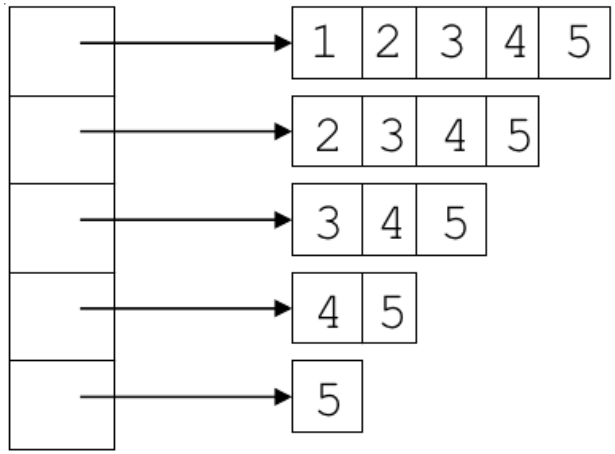
- Un tableau à 2 dimensions est un tableau à 1 dimension dont chaque élément est lui-même un tableau à une dimension
- Un tableau à 3 dimensions est un tableau à 1 dimension dont chaque élément est lui-même un tableau à 2 dimensions
- **Exemple :**
  Soit le tableau `int[][][] x = new int[3][2][5];`



Tableaux asymétriques

- On peut aussi construire un tableau par initialisation de ses éléments

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



- Ou encore, si l'on ne connaît pas les valeurs initiales du tableau `x`

```
int[][] x = new int[5][];
x[0]= new int[5];
x[1]= new int[4];
x[2]= new int[3];
x[3]= new int[2];
x[4]= new int[1];
```

Exemple : plateau du Tic Tac Toe (1/3)

- Le Tic-Tac-Toe se joue sur une grille carrée de 3x3 cases. Deux joueurs s'affrontent. Ils doivent remplir chacun à leur tour une case de la grille avec le symbole qui leur est attribué : 0 ou 1. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale.

- **Algorithme :**
  - remplir aléatoirement la matrice avec l'un des deux entiers : 1 ou 0
  - afficher le plateau
  - découvrir les lignes, colonnes ou diagonales exclusivement composées de 1 ou 0
  - afficher le résultat

Exemple : plateau du Tic Tac Toe (2/3)

```
import java.util.*;

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

    // initialisation du plateau
    Random alea = new Random();
    int[][] plateau = new int[3][3];

    for (int i=0;i<plateau.length;i++)
      for (int j=0;j<plateau[0].length;j++)
      // la methode nextInt(int a) de la classe Random
      //renvoie un entier entre 0 et a-1
      plateau[i][j] = alea.nextInt(2);

    // affichage du plateau
    for (int i=0;i<plateau.length;i++){
      for (int j=0;j<plateau[0].length;j++)
          System.out.print( plateau[i][j] );
      System.out.println();
    }
```

Exemple : plateau du Tic Tac Toe (3/3)

```
  //test lignes
  int[] ligne = null;
  for (int i=0;i<plateau.length;i++){
    ligne = plateau[i];
    if( ligne[0]==ligne[1]  ligne[0]==ligne[2] )
     System.out.println("ligne["+i+"]="+ligne[0]+ligne[1]+ligne[2]);
  }

  //test colonnes
  for (int j=0;j<plateau[0].length;j++){
   if( plateau[0][j]==plateau[1][j]  plateau[0][j]==plateau[2][j]){
    System.out.print("colonne["+j+"]=");
    System.out.println(plateau[0][j]+plateau[1][j]+plateau[2][j]);}
  }

  // test diagonales
  if( plateau[0][0]==plateau[1][1]  plateau[0][0]==plateau[2][2]){
   System.out.print("diagonale 1 =");
   System.out.println(plateau[0][0]+plateau[1][1]+plateau[2][2]);}
  if(plateau[0][2] == plateau[1][1]  plateau[1][1] == plateau[2][0]){
   System.out.print("diagonale 2 =");
   System.out.println(plateau[0][2]+plateau[1][1]+plateau[2][0]);}
 }
}
```

Retour sur la méthode main

- La méthode `main` prend un paramètre de type `String[]`, tableau de chaînes de caractères.
- Ce paramètre permet de transférer des données entre la ligne de commande et le programme java

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

```
>java Transfert un 12 6.54
un
12
6.54
```

args → | "un" | "12" | "6.54" |