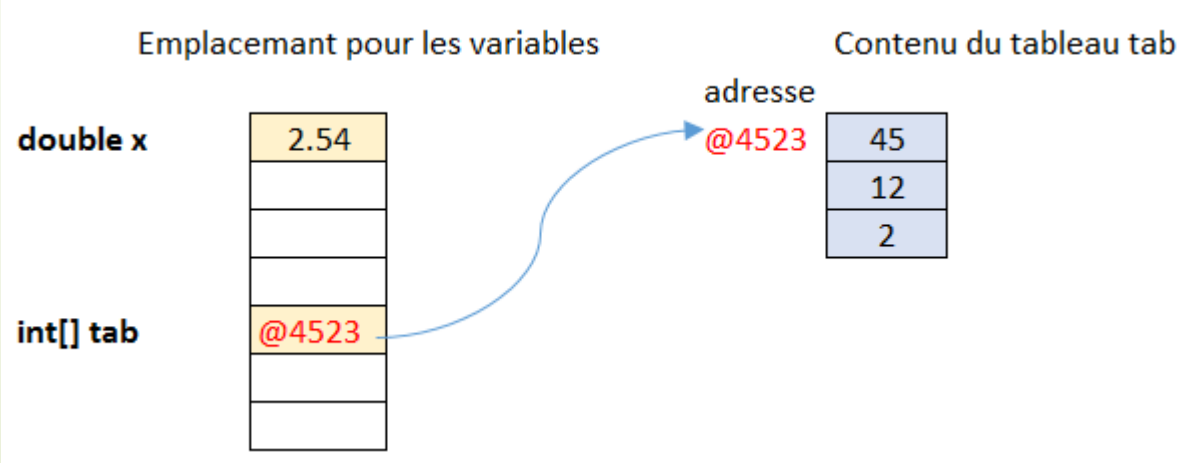
Les tableaux

Catégorie de données en Java (1/2)

- Il existe deux catégories de données en Java
  - les donnés de type primitif : se sont des valeurs élémentaires ou de type simple comme char, byte, int, long, float, double, boolean.
  - Les données de type référence: formées de plusieurs données plus élémentaires comme les tableaux, String, objets, Scanner,etc.

La représentation en mémoire des données en Java (2/2)

- A toute variable correspond un emplacement de stockage en mémoire.
- L'emplacement mémoire d'une variable de type primitif contient sa valeur .
- L'emplacement mémoire d'une variable type référence ne contiennent pas sa valeur, mais une adresse vers celles-ci.



Déclarer un tableau (1/2)

- Comme toute variable, un tableau doit être déclaré.
- Un tableau se déclare comme n'importe quelle variable de type simple, **en ajoutant une paire de crochets entre le type et le nom de la variable** pour indiquer au compilateur que la variable représente un tableau.
- Syntaxe de déclaration d'un tableau

  TypeTableau**[]** nomTableau
- Exemples:

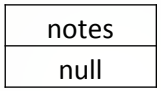
```
double[] notes;
String[] nomEtudiants;
```

- Java autorise aussi de placer les crochets après le nom de la variable.

```
double notes[];
```

Déclarer un tableau (2/2)

- Quand le compilateur rencontre la déclaration d'un tableau comme `double[] notes;` il réserve un espace mémoire portant le nom du tableau (ici notes) qui contient la valeur null pour l'instant.



- La variable notes n'a aucune composante à ce moment.
- L'instruction notes[2] provoque une erreur de type NullPointerException

initialiser un tableau (1/4)

- Il existe deux façons d'initialiser un tableau en Java.

1. Initialiser un tableau sans définir ses éléments.

```
double notes[];//déclaration
notes=new double[10];//initialisation
```

Note: on peut déclarer et initialiser un tableau en une seule instruction.

```
double notes[] = new double[10]; //déclaration et initialisation
```

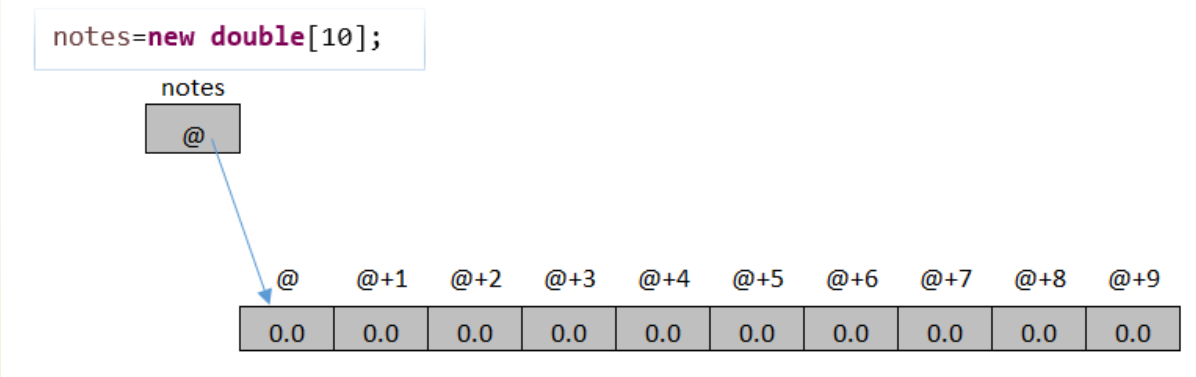
2. Initialiser un tableau en donnant ses éléments lors de la déclaration:

Exemples

```
String[] noms = {"Lynda", "Alain", "Sylvie", "Pascal"};
int[] jour = {1, 2, 3, 4, 5, 6, 7};
```

Initialiser un tableau 2/4

- `double notes[] = new double[10];`
- L'opérateur **new**
  - réserve le nombre de cases mémoires consécutives qu'il est indiqué entre [] (crochets.
  - Initialise des composantes avec les valeurs par défaut.



- La taille de cet espace est non modifiable. On dit que les tableaux sont de taille fixe

Initialiser un tableau 3/4

- L'initialisation d'un tableau avec l'opérateur **new** remplit les cases du tableau avec des valeurs par défaut.

Type	**Valeur par défaut**
int,byte, short,long	0
double, float	0.0
boolean	false
char	Caractère nul '\u0000'
Objet quelconque	null

Initialiser un tableau 4/4

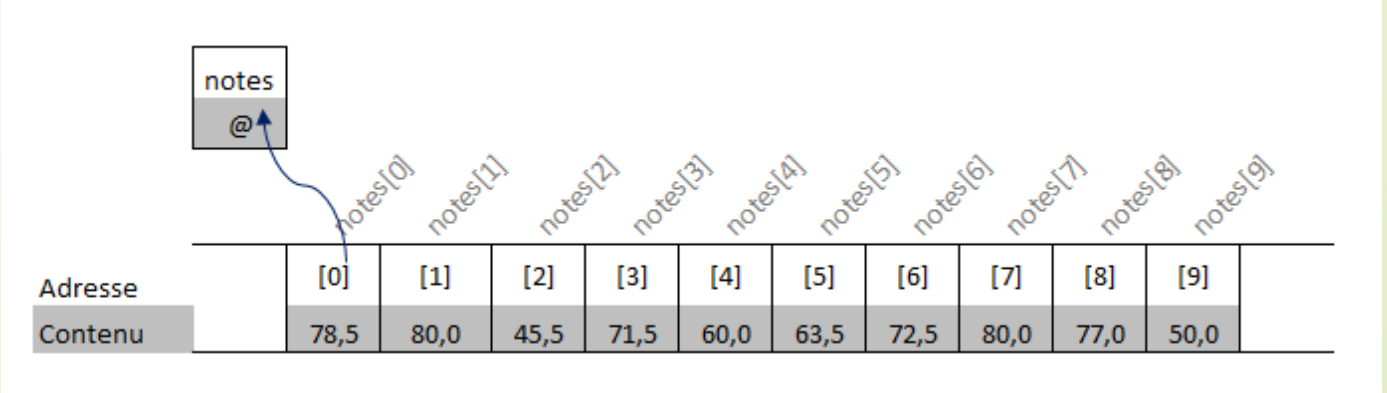
- Une fois que l'opérateur new est exécuté, la taille du tableau est fixée et on ne peut plus la modifier.
- Cependant on peut ne pas fixer la taille du tableau avant exécution et mettre une variable pour la taille entre[](crochets) au lieu d'une valeur. Voir l'exemple ci-dessous. TailleTableau.java

```
Scanner clavier = new Scanner(System.in);
double[] notes;
int nbNotes;
System.out.println("Combien de notes ");
nbNotes = clavier.nextInt();
notes = new double[nbNotes];
```

- Dans l'exemple précédent, l'utilisateur lit la taille du tableau avant de réserver les espaces mémoire par l'opérateur new. La taille du tableau varie alors à chaque exécution.

Accès aux éléments d'un tableau (1/2)

- Chaque case d'un tableau est identifiée par un indice(numéro).
- L'indice de la **première case est 0** (et non 1).
- L'indice de la deuxième case est 1, ainsi de suite
- Pour un tableau de **n éléments**, l'indice de la **dernière case est n-1**



- Pour stocker la valeur 71,5 dans la quatrième case du tableau, on écrit: `notes[3]=71.5;`

Accès aux éléments d'un tableau (2/2)

- Les éléments d'un tableau sont **ordonnés** grâce à leur **indice**, on peut donc les manipuler avec des instructions itératives.
- **Un tableau est un objet** caractérisé par des attributs (données) et des méthodes. L'attribut **length** nous donne la **taille du tableau**.

```
Exemple: notes.length
```

- Exemple parcours du tableau avec une boucle for

```
for (int i = 0; i < notes.length; i++) {
    System.out.print(notes[i] + " ");
}
```

Deuxième type de boucle for pour parcourir un tableau (1/2)

- Java propose depuis la version **1.5** du langage une **nouvelle syntaxe** pour la boucle **for** qui permet de parcourir des tableaux plus simplement que la boucle for classique.
- Exemple pour afficher le tableau des notes

```
for (double valeur : notes) {
     System.out.println(valeur);
}
```

- Cette boucle for **n'utilise pas de compteur** de boucle **ni de test de fin de boucle**. La gestion du compteur est transparente pour le programmeur.
- Fonctionnement: Le tableau est parcouru du premier élément jusqu'au dernier. La valeur de chaque case est enregistrée tour à tour dans la variable valeur.

Deuxième type de boucle for pour parcourir un tableau (2/2)

Limites de cette boucle for

```
for (double valeur : notes) {
        System.out.println(valeur);
    }
```

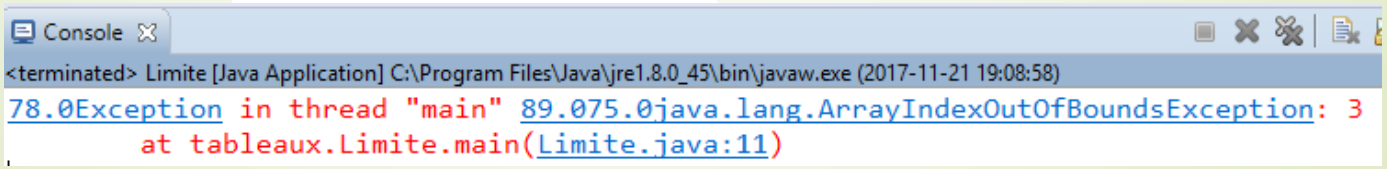
- Elle ne permet pas de modifier le contenu des cases.
- Elle ne permet pas d'itérer sur plus qu'un tableau.
- Il n'est pas possible de récupérer l'élément précédent ni le suivant.
- Elle ne permet pas de parcourir le tableau de la fin vers le début.

Accès aux éléments d'un tableau

- Pour un tableau tab
  - Le premier indice est 0
  - Le dernier indice est **tab.length -1**
- Exemples:
  - `int[] tableau1 = {78, 89, 44, 96, 74};`
    Les indices varient de 0 à 4
  - `double[] notes = new double[10];`
    Les indices varient de 0 à 9
- Si on tente d'accéder à un élément du tableau avec un indice **hors des limites** du tableau, le compilateur Java générera une **erreur** (ou une exception) de type **ArrayIndexOutOfBounds** et le programme arrête.

```
double[] notes = {78, 89, 75};
for (int i = 0; i <= 4; i++) {
    System.out.print(notes[i]);
}
```

indice hors limite



Opérations arithmétiques sur les tableaux

- Les opérations arithmétiques directes sur les tableaux ne sont pas possibles.
- Exemple: addition de 2 tableaux

```
int[] tableau1 = {78, 89, 44, 96, 74};
int[] tableau2 = {85, 56, 89, 78, 94};
int[] tablSomme = new int[5];
```

L'écriture ci-dessous n'est **pas possible** en Java

```
tableSomme = tableau1 + tableau2;
```

- Il faut additionner 2 tableaux élément par élément

```
for (int i = 0; i < tablSomme.length; i++) {
    tablSomme[i] = tableau1[i] + tableau2[i];
}
```

Tableau en argument (1/3)

Comment spécifier un paramètre de type tableau?

- La spécification d'un paramètre de type tableau se fait sans spécifier sa taille: avec des crochets vides[] . On peut obtenir la taille du tableau à l'aide de l'attribut **length**.
- Exemple.

```
3  public class ParametreTableau {
4      public static void main(String[] args) {
5          double[] notes = {78.0, 89.5, 75.0};
6          afficherNotes(notes);
7      }
8      public static void afficherNotes(double[] tabNotes) {
9          for (int i = 0; i < tabNotes.length; i++) {
0              System.out.println("note#" + (i + 1) + " " + tabNotes[i]);
1          }
2      }
3  }
```

crochets vides

Tableau en argument (2/3)

- Lorsqu'on transmet un nom de tableau en argument d'une méthode, on transmet **une copie de la référence au tableau.**
- Cependant, même si la référence est copiée, cette copie pointe toujours vers le même objet (les mêmes cases du tableau). La méthode agit alors directement sur l'objet tableau et non pas une **copie**.
- Autrement dit, si la méthode modifie les éléments du tableau passé en argument, cela aura une influence sur ces éléments même en dehors de la méthode.
- Exemple: ParametreTableau1.java

Voir un exemple complet à la diapositive suivante.

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

    double[] notes = {78.0, 89.5, 75.0};
     initialiserNotes(notes);
}
public static void initialiserNotes(double[] tabNotes) {
       for (int i = 0; i < tabNotes.length; i++) {
              tabNotes[i] = 0;
}
}
```

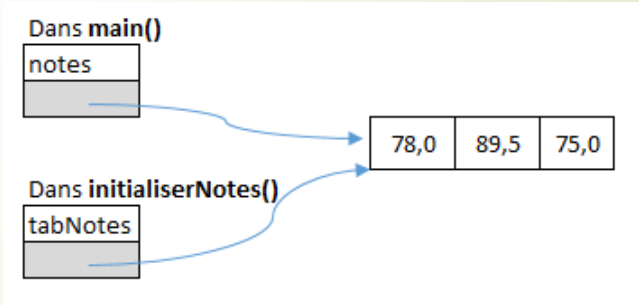


Tableau en argument (3/3)

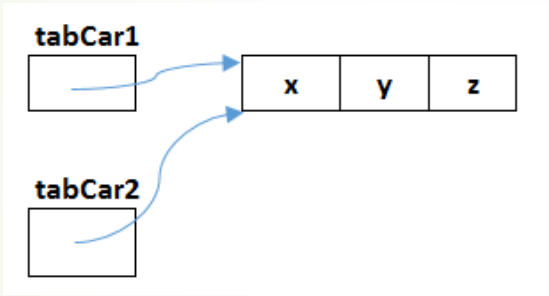
Exemple complet: ParametreTableau2.java

```
3  public class ParametreTableau2 {
4      public static void main(String[] args) {
5          double[] notes = {78.0, 89.5, 75.0};
6          System.out.println("Avant l'appel de initialiserNotes()");
7          afficherNotes(notes);
8          initialiserNotes(notes);
9          System.out.println(
10                 "Après l'appel de initialiserNotes(), les cases du tableau sont modifiées")
11         afficherNotes(notes);
12     }
13     public static void afficherNotes(double[] tabNotes) {
14         for (int i = 0; i < tabNotes.length; i++) {
15             System.out.println("note#" + (i + 1) + " " + tabNotes[i]);
16         }
17     }
18     public static void initialiserNotes(double[] tabNotes) {
19         for (int i = 0; i < tabNotes.length; i++) {
20             tabNotes[i] = 0;
21         }
22     }
23 }
```

```
<terminated> ParametreTableau2 [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (2017-11-21 21:51:45)
Avant l'appel de initialiserNotes()
note#1 78.0
note#2 89.5
note#3 75.0
Après l'appel de initialiserNotes(), les cases du tableau sont modifiées
note#1 0.0
note#2 0.0
note#3 0.0
```

Affectation entre variables de type tableau (1/2)

```
char[] tabCar1 = {'x','y','z'};
char[] tabCar2 ;
tabCar2 = tabCar1;
```



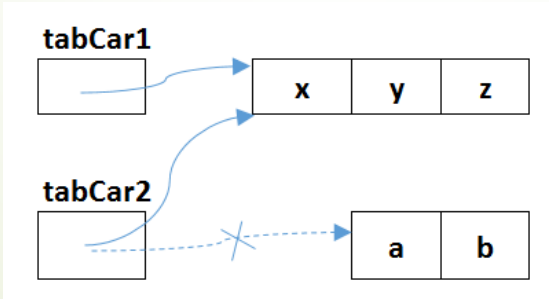
- Les variables tabCar1 et tabCar2 contiennent la même adresse. Elles pointent vers le même emplacement physique de la mémoire. On dit qu'elles **partagent le même espace**.
- Tout changement dans tabCar1 touche tabCar2
  si on écrit, `tabCar1[0]='a';`
  `tabCar2[0]` vaudra également `'a'`
- Verification à l'aide du débogueur. Fichier AffectationTableau.java,

Affectation entre variables de type tableau(2/2)

On peut affecter à un tableau tabCar1 un autre tableau tabCar2 de même type, mais de longueur différente.

```
char[] tabCar1 = {'x','y','z'};
char[] tabCar2= {'a','b'};
tabCar2 = tabCar1;
System.out.println(tabCar2.length);  //affiche 3
```

La longueur de tabCar2 change, elle devient égale à celle de tabCar1, puisque tabCar2 pointe vers le même espace physique que tabCar1.



Comparer des variables de types référence

- L'opérateur **==** ne peut être utilisé pour vérifier si deux tableaux ont le même contenu.
- L'opérateur **==** compare **les adresses mémoires**.

```
char[] tab1 = { 'x', 'y', 'z' };
char[] tab2 = { 'x', 'y', 'z' };
char[] tab3 = tab2;
System.out.println(tab1 == tab2); // affiche false
System.out.println(tab3 == tab2); // affiche true
```

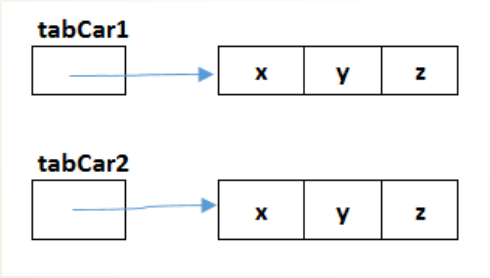
  - tab1 == tab2 est faux. tab1 et tab2 ne pointent pas vers le même emplacement mémoire
  - tab2 == tab3 est vrai. tab2 et tab3 pointent vers le même emplacement mémoire
- Pour comparer si deux tableaux sont égaux en terme de contenu, il faut écrire une méthode qui compare une à une chacune de leurs cases.
- Ceci s'applique pour le type **String** qui est un type référence. Des méthodes ont été implémentées: **equals**() et **equalsIgnoreCase**().

Le clonage de tableau

```
char[] tab1 = { 'x', 'y', 'z' };
```

- Si on veut travailler avec une copie du tableau pour la faire évoluer différemment , on crée un clone avec la méthode clone().

```
1. char[] tabCar1 = {'x','y','z'};
2. char[]tabCar2;
3. tabCar2= tabCar1.clone();
4. tabCar2[0] = 'a';
```



- Les modifications qui seront apportées dans tabCar2 (comme dans la ligne 4), n'affecteront pas le tableau tabCar1 et vis-versa.

Le clonage de tableau

- Attention: La méthode clone() par défaut qui est utilisée avec les tableaux fait une copie de surface (copie bit à bit) , c'est-à-dire, les objets contenus dans le tableau sont copiés par référence.
- Vérifier avec le débogueur le clonage de tableau. Fichier ClonageTableau.java

Exercice1

- Écrivez la méthode **appliquerBonus** de la classe **Exercice1.java** qui ajoute un bonus à toutes les cases d'un tableau de note. La note après application du bonus ne peut dépasser 100.

```
public static int[] appliquerBonus(int[] tabNotes, int  bonus)
```

- Vous ne devez pas modifier le tableau notes, travaillez avec une copie du tableau.
- Voici la méthode main() de la classe Exercice1

```
public static void main(String[] args) {
    int [] notes = { 70, 80, 84, 75, 81, 50, 100 };
    afficher(notes);
    int [] notesBonus = appliquerBonus(notes, 3);
    afficher(notesBonus);
}
```

Échanger deux variables

- Quel énoncé permet d'échanger les deux cases noms[i] et noms[j] d'un tableau?

Énoncé1

```
tab[i]= tab[j];
tab[j]= tab[i];
```

Énoncé 2

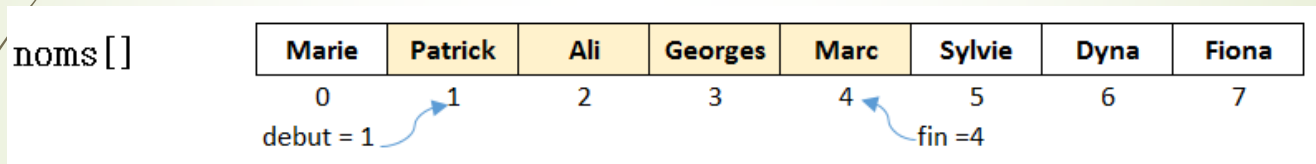
```
tab[i]= tab[j];
temp = tab[i];
tab[j]= temp;
```

Énoncé 3

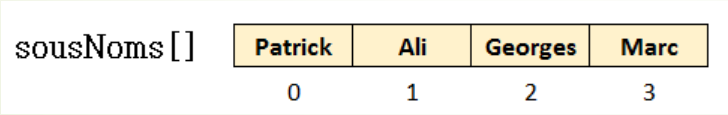
```
temp = tab[i];
tab[i]= tab[j];
tab[j]= temp;
```

Sous-tableau

- On peut copier certains éléments d'un tableau dans un nouveau tableau. Ce dernier peut être considéré comme un sous-tableau.
- Exemple: créer un tableau qui contiendra les éléments du tableau noms à partir de l'indice **debut** jusqu'à l'indice **fin** .



- Voici le contenu du sous-tableau obtenu.



- Quelle sera la taille du nouveau tableau? Exprimez cette taille en fonction des valeurs des variables debut et de fin.

Sous-tableau

- La signature d'une méthode de création d'un tel sous tableau pourrait être

```
public static String[] extraire(String[] tabNoms, int debut, int fin)
```

- Le principe de création est le suivant

1. Créer et initialiser un tableau sousNom[] de la longueur **fin – debut +1**
2. Parcourir le nouveau tableau créé (sousNom) et copier les éléments du tableau d'origine (tabNoms[]) à partir de l'indice **debut**, dans le nouveau tableau.

```
pour i de 0 à sousNoms.length
        sousNoms[i] = noms[debut+i]
fin pour
```

Exercice2

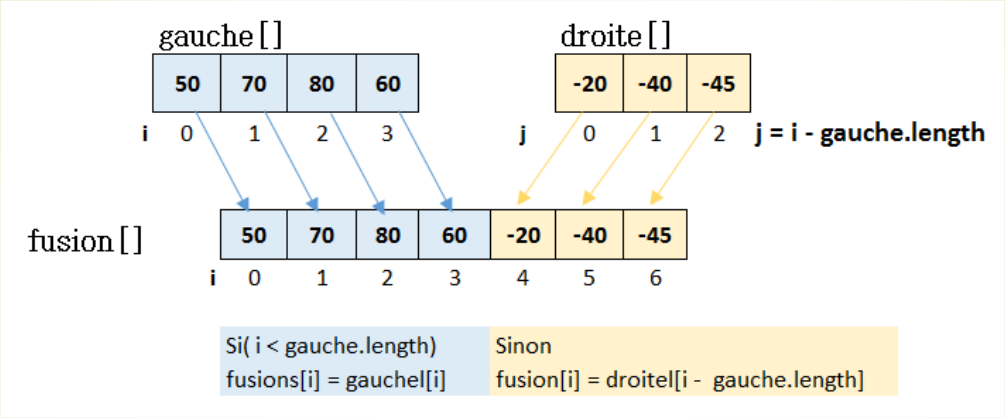
- Complétez la méthode **extraire** de la classe **Exercice2.java**

```
public static String[] extraire(String[] tabNoms, int debut, int
fin) {
    …
}
```

- Ajoutez l'appel de méthode créée pour extraire là partir du tableau **noms** un sous tableau à partir de l'indice **1** à l'indice **4.**

Fusion de tableaux

- On veut fusionner les 2 tableaux **droite** et **gauche**



Le principe

- Créer un tableau **fusion** de taille = gauche.length + droite.length.
- Parcourir le tableau **fusion** et copier les éléments du tableau **gauche**, puis les éléments du tableau **droite**, comme ci-dessous.

```
pour i de 0 à fusion.length
        si ( i < gauche.length)
                fusion[i] = gauche[i]
        Sinon
                fusion[i]= droite[i – gauche.length]
        fin si
fin pour
```