Programmation Objet – Initiation à Java

Letícia SEIXAS PEREIRA

Cours 03 – Éléments de syntaxe

28/10/2019

Éléments de syntaxe

- Tableaux
- Chaînes de caractères
- Les méthodes

Tableaux

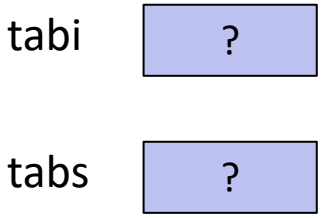
Tableaux à une dimension

Tableaux à une dimension

Déclaration:

- Le type d'un tableau dépend du type de ses éléments:
  - Le type d'un tableau est construit à partir du type de ses éléments suivi d'un paire de crochets.

```
int[] tabi;  // tableau d'entiers
String[] tabs; // tableau de chaînes
```



Tableaux à une dimension

Création:

- Les tableaux sont créés par l'opérateur **new:**
  - L'appel de **new** pour la création d'un tableau est paramétré par le nom du type des éléments suivi d'une paire de crochets contenant la *dimension (i.e. le nombre de cases à créer.*

```
int[] tabi;  // tableau d'entiers
String[] tabs; // tableau de chaînes


tabi = new int[4]; // création d'un tableau de 4 entiers
Tabs = new String[4]; // création d'un tableau de 4 chaînes
```

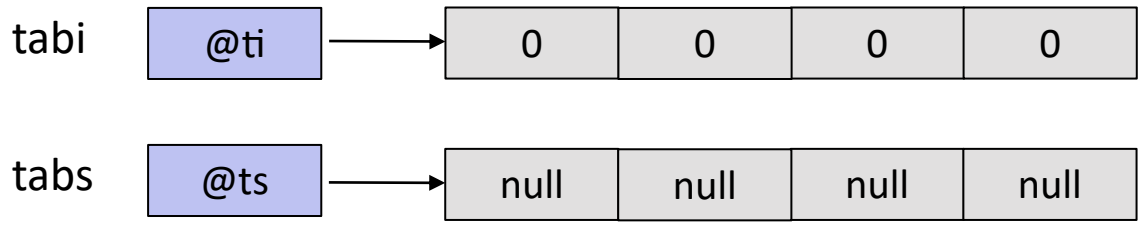
Tableaux à une dimension

Création:

- Les tableaux sont créés par l'opérateur **new:**
  - Les cases d'un tableau ont des valeurs initiales par défaut (*qui dépendent du type de la case).*

```
int[] tabi;
String[] tabs;

tabi = new int[4];
Tabs = new String[4];
```

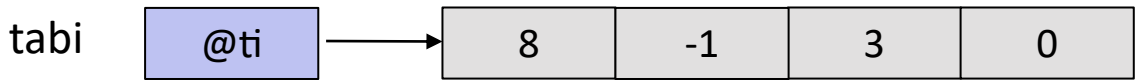


Tableaux à une dimension

Initialisation rapide:

- Un tableau peut être alloué et initialisé directement à sa *déclaration:*

```
int[] tabi = {8, -1, 3, 0};
```



Tableaux à une dimension

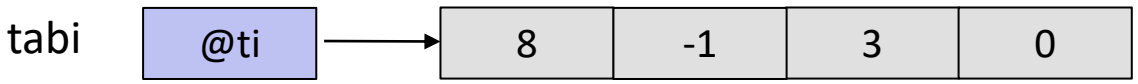
Initialisation rapide:

- Un tableau peut être alloué et initialisé directement à sa *déclaration:*

```
int[] tabi = {8, -1, 3, 0};
```

- Cette notation ne peut pas être utilisée ailleurs qu'à la déclaration. Il s'agit d'un raccourci du code suivant:

```
int[] tabi = new int [4];
tabi[0] = 8;
tabi[1] = -1
tabi[2] = 3;
tabi[3] = 0;
```

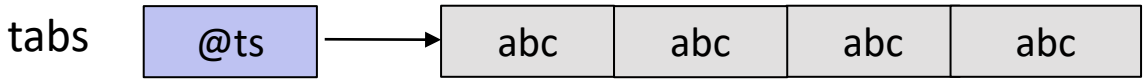


Tableaux à une dimension

Parcours de tableau:

- Un tableau peut être parcouru en *lecture* et *écriture* grâce à un indice variant de 0 (inclus) à la taille du tableau (exclue):

```
String[] tab = new String[4];
for (int i = 0; i < tab.length; i++) {
    tab[i] = "abc";
}
```



Tableaux à une dimension

Parcours de tableau:

- Un tableau peut être parcouru en *lecture* et *écriture* grâce à un indice variant de 0 (inclus) à la taille du tableau (exclue):

```
String[] tabs = new String[4];
for (int i = 0; i < tabs.length; i++) {
    tabs[i] = "abc";
}
for (String s : tabs){
    System.out.println(s);
}
```

Tableaux à une dimension

Exercice d'application: La moyenne

Ecrire une classe *Moyenne* qui prend en arguments une suite de nombres réels positifs ou nuls (correspondant à des notes), stockés dans un attribut **notes** du type tableau d'entier de capacité 6 et calcule la moyenne de ces valeurs

Boite à outils

int i = Integer.parseInt(s) ; *//Permet de convertir la chaîne « s » en un entier.*

float f = Float.parseFloat(s) ; //Permet de convertir la chaîne « s » en un réel à simple précision.

double d = Double.parseDouble(s) ; //Permet de convertir la chaîne « s » en un réel à double précision.

Exemple d'exécution du programme :

```
> java Moyenne 9.6 9.7 10 9.8 9.2 9.9
La moyenne est : 9.7
```

Tableaux multidimensionnels

Tableaux multidimensionnels

- Les tableaux multidimensionnels s'organisent sur des lignes et des colonnes:
  - Le croisement d'une ligne et d'une colonne détermine l'élément du tableau.

Tableaux multidimensionnels

- Les tableaux multidimensionnels s'organisent sur des lignes et des colonnes:
  - Le croisement d'une ligne et d'une colonne détermine l'élément du tableau.
- Déclaration:
  - Crée une référence sur un tableau de type "int" à deux dimensions.

```
<type> [][] <identificateur>;
<type> <identificateur> [][];
```

```
double [][] notes;
double notes [][];
```

Tableaux multidimensionnels

- Les tableaux multidimensionnels s'organisent sur des lignes et des colonnes:
  - Le croisement d'une ligne et d'une colonne détermine l'élément du tableau.
- Déclaration:
  - Crée une référence sur un tableau de type "int" à deux dimensions.

```
<type> [][] <identificateur>;
<type> <identificateur> [][];
```

```
double [][] notes;
double notes [][];
```

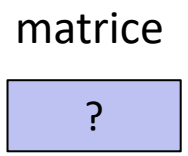
- Création:

```
int matrice[][] = new int [3][3]; // 3 lignes de 3 cases (donc 3 colonnes)
```

Tableaux multidimensionnels

Déclaration:

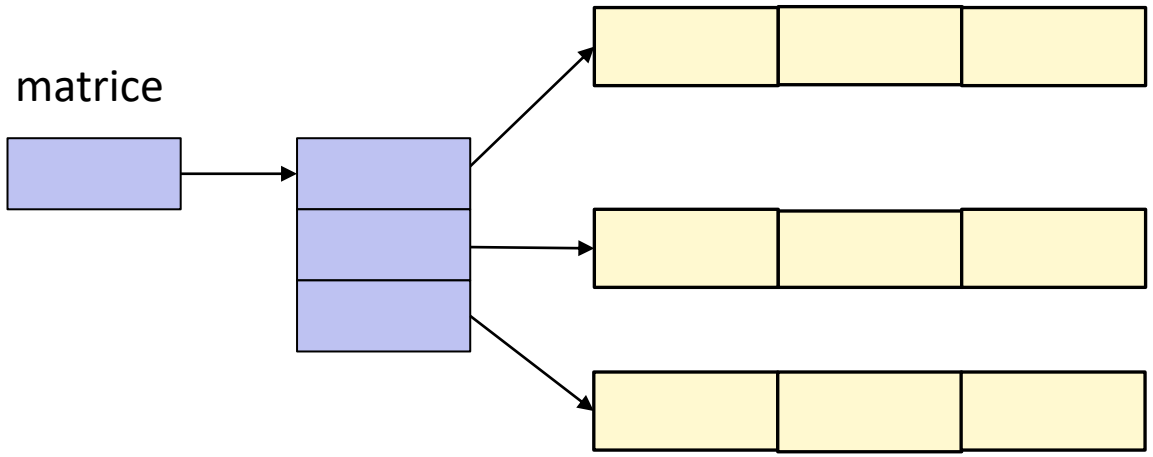
```
int matrice[][];
```



Tableaux multidimensionnels

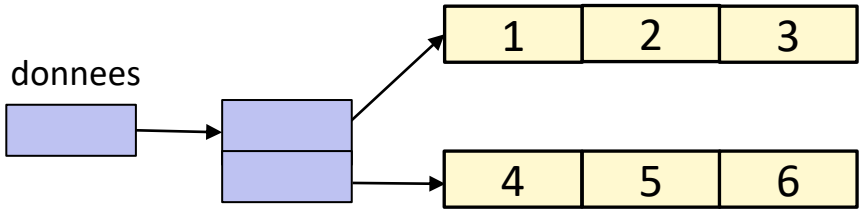
Création:

```
int matrice[][];
matrice = new int[3][3];
```



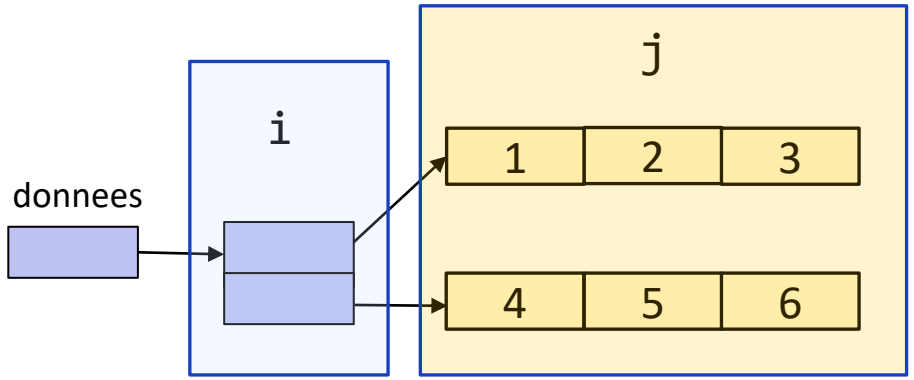
Tableaux multidimensionnels – Manipulation et accès

```
int donnees[][] = {{1,2,3},{4,5,6}};
```



Tableaux multidimensionnels – Manipulation et accès

```
int donnees[][] = {{1,2,3},{4,5,6}};
int i;
int j;
for ( i = 0; i < donnees.length; i++)
    for (j = 0; j < donnees[i].length; j++)
        System.out.println ("donnees["+ i +"]["+ j +"]=" + donnees[i][j]);
```



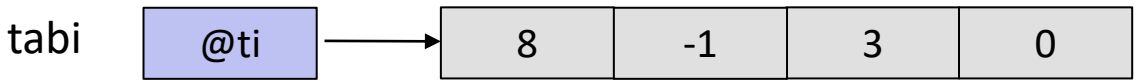
Tableaux multidimensionnels

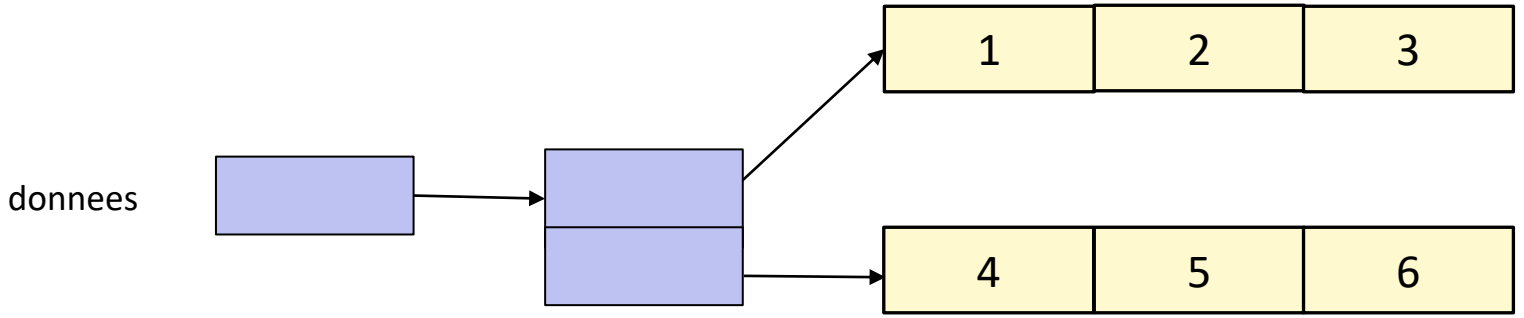
- Java permet la création et la manipulation de tableaux de plusieurs dimensions.
- Il faut juste mettre autant de [] que le nombre de dimensions.

Exemple:

```
int[][][] troisD = { { {1, 2, 3},   { 4, 5, 6},   { 7, 8, 9} },
                     { {10, 11, 12}, {13, 14, 15}, {16, 17, 18} },
                     { {19, 20, 21}, {22, 23, 24}, {25, 26, 27} } };
```

Tableaux multidimensionnels – Manipulation et accès



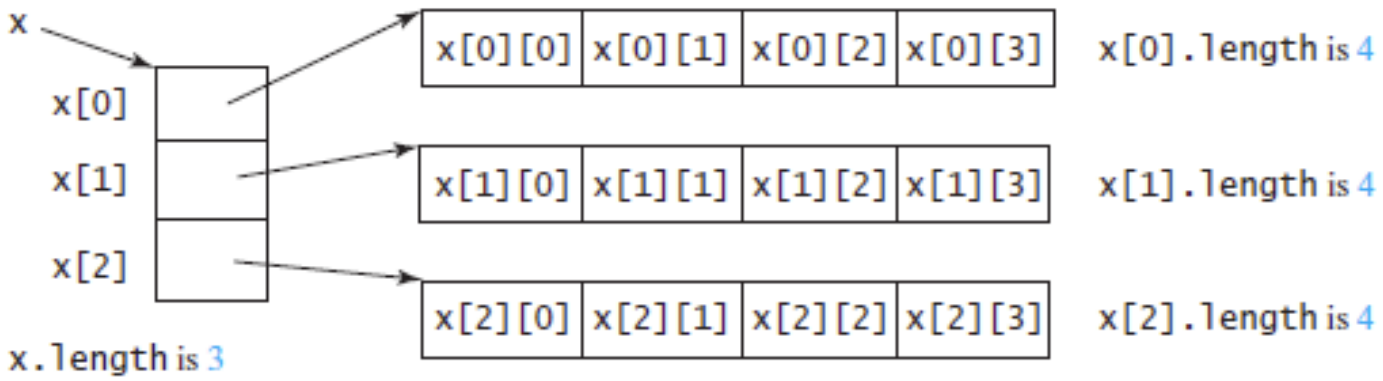


Tableaux multidimensionnels

Exercice d'application:

Donnez un exemple de déclaration, initialisation et affichage d'un tableau 3x4 (*3 lignes et 4 colonnes)*.

- Le contenu de chaque élément du tableau sera la somme de sa ligne avec sa colonne.



Les méthodes

Les méthodes

- Le principal avantage des méthodes et de pouvoir factoriser le code;
- Java offre plusieurs méthodes prédéfinies (natives) organisés dans les classes;

Les méthodes

Méthodes natives : Exemple de méthodes concernant les chaînes de caractères.

- La méthode toLowerCase() permet de transformer tout caractère alphabétique en son équivalent minuscule.

public String toLowerCase () {…}

- La méthode toUpperCase() est simple. Elle transforme donc une chaîne de caractères en capitales.

public String toUpperCase () {…}

- La méthode length() renvoie la longueur d'une chaîne de caractères (en comptant les espaces).

public int length () {…}

- La méthode equals() permet de vérifier (donc de tester) si deux chaînes de caractères sont identiques.

public boolean equals (String s) {…}

Les méthodes

Méthodes natives : Exemple de méthodes concernant les chaînes de caractères.

- Le résultat de la méthode charAt() sera un caractère : il s'agit d'une méthode d'extraction de caractère. Le premier caractère est d'indice 0 et elle prend en argument des entiers.

public char charAt (int pos) {…}

- La méthode substring() extrait une partie d'une chaîne de caractères.

public String substring (int debut) {…}

public String substring (int debut, int fin) {…}

```
String chaine = new String("Master MIASHS 2018");

String chaine2 = chaine.substring(7, 12);
//Permet d'extraire "MIASH"

String chaine3 = chaine.substring(7);
//Permet d'extraire « MIASHS 2018"
```

Les méthodes

- Le principal avantage des méthodes et de pouvoir factoriser le code;
- Java offre plusieurs méthodes prédéfinies (natives) organisés dans les classes;

Les méthodes

Appel et retour de fonction

- L'appel de fonction consiste à invoquer une portion de code identifiée par une signature;
- La signature est donnée par un *type de retour,* un *nom* et une séquence de *paramètres;*

```
int somme (int a, int b){
    //...
}
```

Les méthodes

Appel et retour de fonction

- L'appel de fonction consiste à invoquer une portion de code identifiée par une signature;
- La signature est donnée par un *type de retour,* un *nom* et une séquence de *paramètres;*
  - Un *paramètre* correspond à un nom et à un type et s'utilise comme une variable locale.

```
int somme (int a, int b){
    int c = a + b;
    //...
}
```

Les méthodes

Appel et retour de fonction

- L'appel de fonction consiste à invoquer une portion de code identifiée par une signature;
- La signature est donnée par un *type de retour,* un *nom* et une séquence de *paramètres;*
  - Un *paramètre* correspond à un nom et à un type et s'utilise comme une variable locale.
- La instruction **return** prend en paramètre une valeur à renvoyer en résultat et interrompt l'exécution de la fonction.

```
int somme (int a, int b){
    int c = a + b;
    return c;
}
```

Les méthodes

Appel et retour de fonction

- L'appel de fonction consiste à invoquer une portion de code identifiée par une signature;
- La signature est donnée par un *type de retour,* un *nom* et une séquence de *paramètres;*
  - Un *paramètre* correspond à un nom et à un type et s'utilise comme une variable locale.
- Quand la fonction ne renvoie pas de valeur (type **void)**, l'instruction **return** ne prend pas de paramètre.

```
void somme (int a, int b){
    int c = a + b;
    System.out.println("La somme de a + b est:" + c);
    return; //superflu
}
```

Les méthodes

Appel et retour de fonction : Exemple

```
public class Somme{

    public static int somme(int a, int b){

        return a+b;

    }









}
```

Les méthodes

Appel et retour de fonction : Exemple

```
public class Somme{

    public static int somme(int a, int b){

        return a+b;

    }

    public static void main (String[] args){

        int s = somme(15,20);

        System.out.println(s);
    }
}
```

Les méthodes

Appel et retour de fonction : Exemple

```
public class Somme{

    public static int somme(int a, int b){

        return a+b;

    }

    public static void main (String[] args){

        int s = somme(15,20);

        System.out.println(s);
    }
}
```

```
public class Somme{

    public static void somme(int a, int b){

         System.out.println(a+b);

    }

    public static void main (String[] args){

        somme(15,20);

    }
}
```

Les méthodes

Appel et retour de fonction : Exemple

```
public class Somme{

    public static int somme(int a, int b){

        return a+b;

    }

    public static void main (String[] args){

        int s = somme(15,20);

        System.out.println(s);
    }
}
```

```
public class Somme{

    public static void somme(int a, int b){

         System.out.println(a+b);

    }

    public static void main (String[] args){

        somme(15,20);

    }
}
```

Si vous placez une de vos méthodes à l'intérieur de la méthode main ou à l'extérieur de votre classe, le programme ne compilera pas.

Les méthodes

La surcharge des méthodes

Les méthodes

La surcharge des méthodes:

- En Java il est possible de déclarer deux ou plusieurs méthodes avec le même nom **à condition que:**
  - Les types et/ou le nombre de paramètres soient différents.

```
int somme (int a, int b){
    //...
}
```

Les méthodes

La surcharge des méthodes:

- En Java il est possible de déclarer deux ou plusieurs méthodes avec le même nom **à condition que:**
  - Les types et/ou le nombre de paramètres soient différents.

```
int somme (int a, int b){
    //...
}
```

```
int somme (double a, double b){
    //...
}
```

```
int somme (int a, int b, int c){
    //...
}
```

Les méthodes

Exercice d'application : Créer une méthode

- Nous allons reprendre l'exercice de la moyenne. On vous demande de factoriser le code en répartissant sur 2 méthodes:
  - La première fait le calcul de la moyenne;
  - La dernière permet l'affichage des résultats.

```
public class Somme{

     public static int somme(int a, int b){

          return a+b;

     }

     public static void main (String[] args){

          int s = somme(15,20);

          System.out.println(s;
     }
}
```