

# **Java**

## **Introduction**

### **aux méthodes**

**Préparé par Larbi Hassouni**

# Sommaire:

- Intérêt des méthodes
- Types des méthodes
- Déclaration d'une méthode
- Modificateurs d'accès
- Type de retour (Mot clé void)
- Nom d'une méthode
- Liste des paramètres
- Corps d'une méthode
- Variables locales
- Instruction return
- Invocation d'une méthode
- Commentaires d'une méthode

# Structure simplifiée d'une classe

```
[public] class UneClasse{
```

```
//Constante membre d'instance
```

```
[modificateur] final <type> CONST_INSTANCE [= <valeur>];
```

```
//Constante membre statique ou de classe
```

```
[modificateur] static final >type> CONST_CLASSE = <valeur>;
```

```
//Variable membre d'instance
```

```
[modificateur] <type> variableInstance [=<valeur>];
```

```
//Variable membre statique ou de classe
```

```
[modificateur] static <type> variableClasse [=<valeur>];
```

```
//Méthode membre d'instance
```

```
[modificateur] [type] methodeInstance{  
.....  
}
```

```
//Méthode membre statique ou de classe
```

```
[modificateur] static <type> methodeClasse{  
.....  
}
```

```
}
```

# Intérêt et types des méthodes

- ❑ Une méthode est l'équivalent d'une fonction en C ou fonction/ procédure en pascal
- ❑ Son intérêt est :
  - *Permet de factoriser du code*
  - *permet de structurer le code*
  - *Peut servir de « sous programmes utilitaires » aux autres méthodes de la classe*
  - *...etc*

# Deux types de méthodes

On distingue deux types de méthodes

- **Méthodes d'instances** : *Opèrent sur les objets*
- **Méthodes statiques ou de classe** : *Opèrent uniquement sur les attributs statiques ou de classes*

# Déclaration d'une méthode

- ❑ « Une déclaration de méthode définit du code exécutable qui peut être invoqué, en passant éventuellement un nombre fixé de valeurs comme arguments »

## ❑ Déclaration d'une méthode

```
[Modificateur d'accès] [static] <typeRetour> nomMethode ([ <liste de paramètres>] ) {  
  
    <corps de la méthode>  
  
}
```

### ▪ *exemple*

```
static double max(double a, double b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

Déclaration d'une fonction statique de nom max, qui reçoit deux arguments de type double et retourne une valeur de type double

# Modificateurs d'accès d'une méthode

- ❑ **Modificateur d'accès** : permet de définir les endroits à partir desquels on peut accéder à la méthode

**[Modificateur d'accés] = public | private | protected | -**

- **public** : la méthode peut être invoquée à partir du code de n'importe quelle autre classe
- **private** : la méthode peut être invoquée uniquement à partir du code de la classe qui la définit
- **protected** : voir héritage
- **aucun modificateur** : la méthode peut être invoquée à partir du code de n'importe quelle classe du même package
  
- Nous reviendrons en détail sur ces modificateurs lors de l'étude de l'encapsulation

# Type de la valeur renvoyée par une méthode

## □ <typeRetour>

- Quand la méthode renvoie une valeur, il doit indiquer le type de la valeur renvoyée (type primitif, nom d'une classe, tableau, ...)

### Exemple

- ✓ public **double** max(double a, double b)  
// retourne une valeur **double**
- ✓ public **Point** plusProche(Point[ ] plan)  
// retourne un objet instance de la classe Point
- ✓ Public static **int[ ]** premiers(int n)  
// retourne un tableau d'éléments de type **int**

- Quand la méthode ne renvoie pas de valeur, il doit être **void**

### Exemple

- ✓ public static **void** trierTableau (int[ ] a)
- ✓ public **void** afficherTableau (int[ ] a)
- ✓ public static **void** main (String[ ] args)



# Liste des paramètres d'une méthode

## □ <liste de paramètres>

- une suite de couples <type> <identificateur> séparés par des virgules

### Exemples:

- ✓ public double max(**double a, double b**)  
//reçoit deux paramètres de type double
- ✓ public static int[ ] fusionnerTableau(**int[ ] tab1, int[ ] tab2**)  
//reçoit deux paramètres qui sont des tableaux d'entiers
- ✓ private double distance(**Point p**)  
// reçoit un paramètre objet instance de la classe Point

- vide si la méthode n'a pas de paramètres

### Exemples:

- ✓ public void lireEntier( )  
// ne reçoit aucun paramètre
- ✓ public static int[ ] nombreCube ( )  
// ne reçoit aucun paramètre

# Corps d'une méthode

## □ <corps de la méthode>

- C'est une suite de déclarations de variables locales et d'instructions
- si le type de retour de la méthode est différent de void, son corps doit contenir au moins une instruction **return *expression*** où ***expression*** s'évalue en une valeur compatible avec le type de retour déclaré.

## Exemples:

```
public double max(double a, double b) {  
    double resultat;  
    if (a > b)  
        resultat = a;  
    else  
        resultat = b;  
    return resultat;  
}
```

```
public double min(double a, double b) {  
    if (a < b)  
        return a;  
    else  
        return b;  
}
```

## Fin de l'exécution d'une méthode : instruction return

- ❑ *si la méthode à un type de retour le corps de la méthode doit contenir **au moins** une instruction **return expression** ...*

```
public static boolean contient(int[] tab, int d) {
    boolean trouve = false;
    for(int i = 0; i < tab.length(); i++) {
        if (tab[i] == d){
            trouve = true;
            break;
        }
        i++;
    }
    return trouve
}
```

```
public static boolean contient(int[] tab, int d)
{
    for(int i = 0; i < tab.length(); i++) {
        if (tab[i] == d)
            return true;
        i++;
    }
    return false;
}
```

- *Possibilité d'avoir plusieurs instructions **return***

- Lorsqu'une instruction return est exécutée on retourne immédiatement au programme appelant
- Les instructions suivant le return dans le corps de la méthode ne sont pas exécutées

## Fin de l'exécution d'une méthode

- **return** sert aussi à sortir d'une méthode sans renvoyer de valeur (méthode ayant **void** comme type retour)

```
static void afficherPosition(int[] tab, int d) {  
    for (int i = 0; i < tab.length; i++)  
        if (tab[i] == d){  
            System.out.println("La position de " + d + " est " + i);  
            return;  
        }  
    System.out.println(val + " n'est pas présente dans le tableau");  
}
```

Sortie d'une méthode par  
l'intermédiaire de return

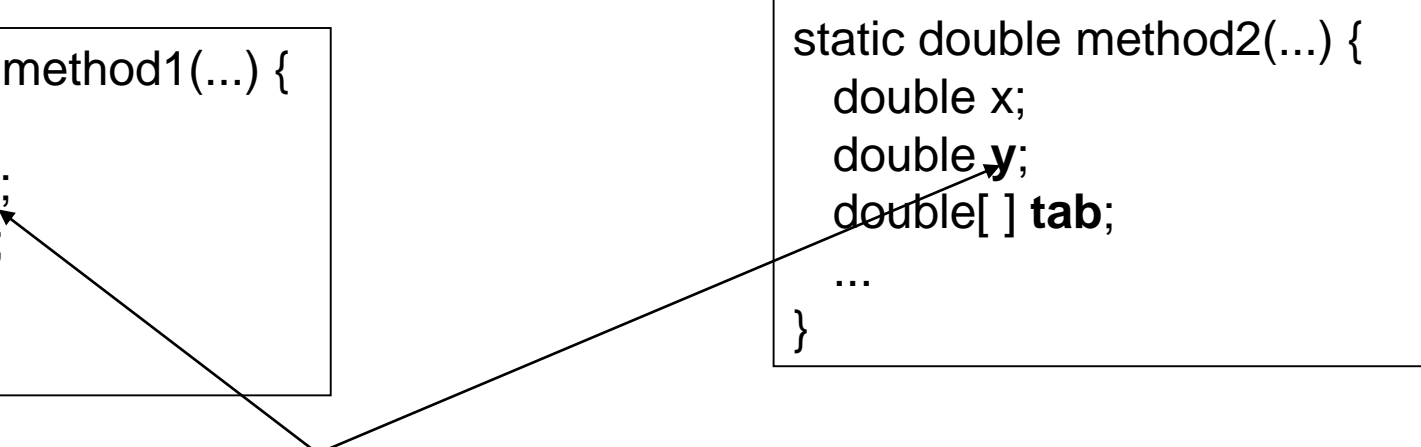
Sortie d'une méthode parce  
qu'on est arrivé à la dernière  
instruction

# Variables locales d'une méthode

- ❑ *Les variables locales sont des variables déclarées à l'intérieur d'une méthode*
  - elles conservent les données qui sont manipulées par la méthode
  - elles ne sont accessibles que dans le bloc dans lequel elles ont été déclarées, et leur valeur est perdue lorsque la méthode termine son exécution

```
static void method1(...) {  
    int i;  
    double y;  
    int[ ] tab;  
    ...  
}
```

```
static double method2(...) {  
    double x;  
    double y;  
    double[ ] tab;  
    ...  
}
```



- **Possibilité d'utiliser le même identificateur dans deux méthodes distinctes**
- **pas de conflit, c'est la déclaration locale qui est utilisé dans le corps de la méthode**

Remarque : Les variables locales à une méthode ne sont pas initialisées par défaut, contrairement aux variables membres

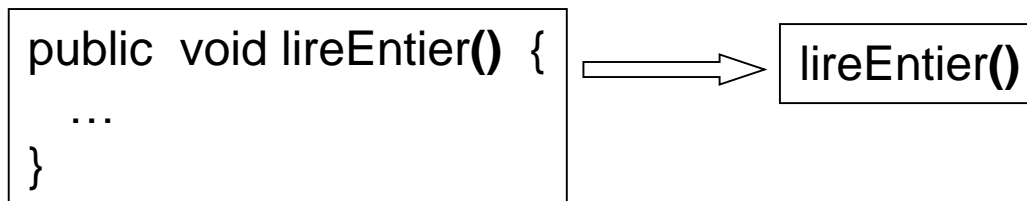
# Invocation d'une méthode

□ Appel : *nomMethode(<liste de paramètres effectifs>)*

▪ <liste de paramètres effectifs>

✓ **Vide** si liste de paramètres vide

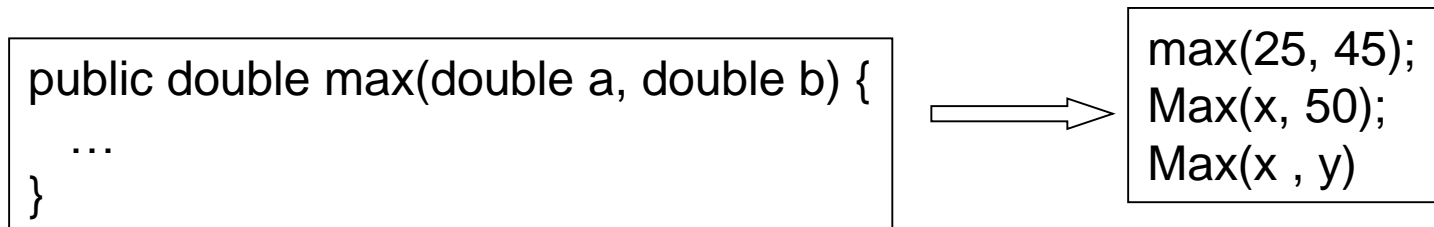
```
public void lireEntier() {  
    ...  
}
```



lireEntier()

✓ **Liste d'expressions** dont le nombre et le type correspond au nombre et au type des paramètres de la méthode

```
public double max(double a, double b) {  
    ...  
}
```



max(25, 45);  
Max(x, 50);  
Max(x, y)

# Invocation d'une méthode : Passage de paramètres

- ❑ Le passage de paramètres lors de l'appel d'une méthode est un passage par valeur.
  - *À l'exécution le paramètre formel défini dans la signature de la méthode correspond à une variable locale au bloc de la méthode*
  - *Elle est initialisée avec la valeur de l'expression définie par le paramètre effectif.*

## Exemples:

```
//méthode qui permet d'échanger deux valeurs
public void echanger(int a, int b) {
    int t;
    t = a; a = b; b = t
}
```

//utilisation de la méthode echanger

```
int x = 10, y = 20;
```

```
echanger(x, y);
```

```
System.out.println (« x = « + x + « \ty = « + y);
```

Sortie :

```
X = 10 y = 20
```

Lors de l'appel de la méthode echanger le système copie la valeur de x dans a, celle de y dans b. Puis echanger opère sur les valeurs de a et b et les échange, alors que celles de x et y restent inchangées

# Invocation d'une méthode

## Passage de paramètres de type objet

```
// Le passage d'objets à une méthodes peut avoir  
// un effet différent de celui qu'on espère
```

```
class Letter {  
    char c;  
}  
  
public class PassObject {  
    static void f(Letter y) {  
        y.c = 'z';  
    }  
  
    public static void main(String[] args) {  
        Letter x = new Letter();  
        x.c = 'a';  
        System.out.println("1: x.c: " + x.c);  
        f(x);  
        System.out.println("2: x.c: " + x.c);  
    }  
} ///:~
```

Sortie du programme:

```
1: x.c: a  
2: x.c: z
```

L'objet original est modifié  
après l'appel de la méthode

Lorsqu'on passe un objet  
comme paramètre à une  
méthode, on copie sa  
référence dans le paramètre.  
Par conséquent si la méthode  
modifie l'objet référencé par le  
paramètre, l'objet initial est  
également modifié.



# L'ordre de déclaration des méthodes N'a pas d'effet sur l'Invocation d'une méthode

- ❑ Toute méthode statique d'une classe peut être invoquée depuis n'importe quelle autre méthode statique de la classe
- ❑ *L'ordre de déclaration des méthodes n'a pas d'importance*

```
public class A {  
    → static void methode1() {  
        ...  
    }  
  
    static void methode2() {  
        → methode1();  
        → methode3();  
    }  
  
    → static void methode3() {  
        ...  
    }  
}
```

Pour invoquer une méthode statique d'une autre classe il faut la préfixer par **NomClasse**.

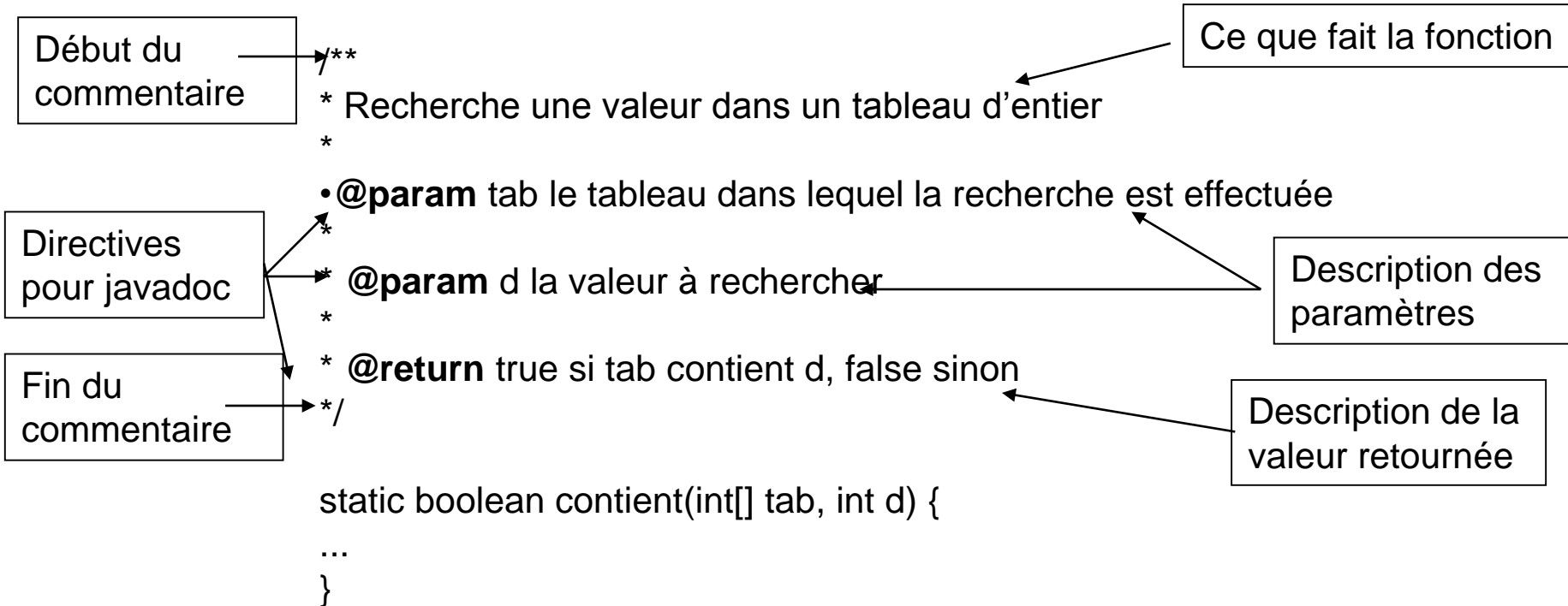
**NomClasse.methodeStatique**  
ou  
**nomObjet.methodeStatique**

Pour invoquer une méthode non statique d'une autre classe il faut la préfixer par **nomObjet**.

**nomObjet.methodeNonStatique**  
~~**NomClasse.methodeNonStatique**~~

# Commentaires d'une méthode

- ❑ Il est recommandé que toute déclaration de méthode doit **TOUJOURS** être précédée de son commentaire documentant (exploité par l'outil javadoc)



# Méthodes ayant un nombre d'arguments variable

- ♠ Quelquefois il peut être commode d'écrire une méthode avec un nombre variable d'arguments
- ♠ L'exemple typique est la méthode *printf* du langage C qui affiche des arguments selon un format d'affichage donné en premier argument
- ♠ Depuis le JDK 5.0, c'est possible en Java

# Syntaxe pour arguments variables

```
Type nomMethode (type param1, type param2, type... paramn){  
    //corp de la méthode  
}
```

Exemples:

```
double max(double x1, double... xn){
```

```
.....
```

```
}
```

```
int min(int x1, int x2, int... xn){
```

```
.....
```

```
}
```

```
public class ArgsVariables{
```

```
private static int sumVarArgs(int... nbs){  
    int s = 0;  
    for(int i = 0 ; i<nbs.length; i++)  
        s+= nbs[i];  
    return s;  
}
```

```
public static void main(String[] args){  
    int a, b, c;  
    int[ ]d = {10,20, 30, 40};  
    a = sumVarArgs(10, 20, 30, 40);  
    b = sumVarArgs(d);  
    c = sumVarArgs(10, 20);  
    System.out.println("a = "+a + "b = " + b + "c = " +c);  
}  
}
```

# Traduction du compilateur

♠ Le compilateur traduit ce type spécial par un type tableau :

**m(int p1, String... params)**

est traduit par

**m(int p1, String[ ] params)**

♠ Le code de la méthode peut utiliser **params** comme si c'était un tableau (boucle **for**, affectation, etc.)

# Remarque

On peut passer un tableau en dernier argument ;

les éléments du tableau seront considérés  
comme la liste d'arguments de taille variable  
attendue par la méthode