

Méthodes statiques (introduction)

Avertissement : ce cours ne présente qu'une version « édulcorée » des méthodes.
Ne sont abordées que les méthodes statiques qui pour le moment seront vues comme de simples fonctions.

Philippe Genoud
Philippe.Genoud@imag.fr

dernière modification : 13/12/2023 01:32



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Méthodes

- Méthodes ⇔ fonctions / procédures
 - Pour factoriser du code
 - Pour structurer le code
 - Pour servir de « sous programmes utilitaires » aux autres méthodes de la classe
 - ...
- En java plusieurs types de méthodes
 - Opérations sur les objets (cf. envois de messages)
 - Opérations statiques (méthodes de classe)
 - exemples
 - `Math.random();`
 - `Integer.parseInt();`
- Pour le moment nous ne nous intéresserons qu'au second type

Méthodes statiques - déclaration

- « Une déclaration de méthode définit du code exécutable qui peut être invoqué, en passant éventuellement un nombre fixé de valeurs comme arguments » *The Java Language Specification* J. Gosling, B Joy, G. Steel, G. Bracha
- Déclaration d'une méthode statique

```
static <typeRetour> nomMethode( <liste de paramètres> ) {  
    <corps de la méthode>  
}
```

- exemple

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

Signature de la méthode



Méthodes statiques - déclaration

- « Une déclaration de méthode définit du code exécutable qui peut être invoqué, en passant éventuellement un nombre fixé de valeurs comme arguments » The Java Language Specification J. Gosling, B Joy, G. Steel, G. Bracha
- Déclaration d'une méthode

```
static <typeRetour> nomMethode( <liste de paramètres> ) {  
    <corps de la méthode>  
}
```

- Exemple

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

Signature de la méthode



```
static double min(double a, double b) {  
    return (a<b)?a:b ;  
}
```

Expression conditionnelle

Méthodes statiques - Déclaration

```
static <typeRetour> nomMethode( <liste de paramètres> ) {  
    <corps de la méthode>  
}
```

- **<typeRetour>**

- Quand la méthode renvoie une valeur (fonction) indique le type de la valeur renvoyée (type simple ou nom d'une classe)

```
static double min(double a, double b) // renvoie un nombre double précision  
static int[] premiers(int n) // renvoie un tableau d'entiers  
static Color getColor() // renvoie une référence d'objet Color
```

- **void** si la méthode ne renvoie pas de valeur (procédure)

```
static void afficher(double[][] m)
```

Méthodes statiques - Déclaration

```
static <typeRetour> nomMethode( <Liste de paramètres> ) {  
    <corps de la méthode>  
}
```

- <liste de paramètres>

- vide si la méthode n'a pas de paramètres

```
int lireEntier()  
void afficher()
```

- une suite de couples type identificateur séparés par des virgules

```
double min(double a, double b)  
int min(int[] tab)  
void setColor(Color c)
```

Méthodes statiques - Déclaration

```
static <typeRetour> nomMethode( <liste de paramètres> ) {  
    <corps de la méthode>  
}
```

- **<corps de la méthode>**

- suite de déclarations de variables locales et d'instructions
- si la méthode à un type de retour le corps de la méthode doit contenir **au moins** une instruction **return expression** où **expression** délivre une valeur compatible avec le type de retour déclaré.

```
static double min(double a, double b) {  
    double vMin; ← Variable locale  
    if (a < b)  
        vMin = a;  
    else  
        vMin = b;  
    return vMin; ← Instruction de retour  
}
```

Méthodes statiques - Déclaration

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

```
static boolean contient(int[] tab, int val) {  
    boolean trouve = false;  
    int i = 0;  
    while ((i < tab.length) && (! trouve)) {  
        if (tab[i] == val){  
            trouve = true;  
        }  
        i++;  
    }  
    return trouve;  
}
```




- Possibilité d'avoir plusieurs instructions **return**
- Lorsqu'une instruction **return** est exécutée retour au programme appelant
 - Les instructions suivant le **return** dans le corps de la méthode ne sont pas exécutées

```
for (int i = 0; i < tab.length; i++){  
    if (tab[i] == val)  
        return true;  
}  
return false;
```


Méthodes statiques - Déclaration

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

```
static void afficherPosition(int[] tab, int val) {  
    boolean trouve = false;  
    int i = 0;  
    while ((i < tab.length) && (tab[i] != val)) {  
        i++;  
    }  
    if (i == tab.length) {  
        System.out.println(val + " n'est pas présente dans le tableau");  
    }  
    else {  
        System.out.println("La position de " + val + " est " + i);  
    }  
}
```



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

Méthodes statiques - Déclaration

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

```
static void methode1(...) {  
    int i;  
    double y;  
    int[] tab;  
    ...  
}
```

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

```
static double methode2(...) {  
    double x;  
    double y;  
    double[] tab;  
    ...  
}
```

Méthodes statiques - Déclaration

- Toute déclaration de méthode doit **TOUJOURS** être précédée de son commentaire documentant (exploité par l'outil javadoc)

Ce que fait la méthode

```
/**
 * Recherche un valeur dans un tableau d'entiers
 *
 * @param tab le tableau dans lequel la recherche est effectuée
 * @param val la valeur à rechercher
 *
 * @return true si tab contient val, false sinon
 */
static boolean contient(int[] tab, int val) {
    ...
}
```

directives pour
l'outil javadoc

@param

Description de chaque
paramètre

@return

Explication de la valeur
retournée

Méthodes statiques - invocation

- Déclaration

```
static <typeRetour> nomMethode( <liste de paramètres> ) {  
    <corps de la méthode>  
}
```

- Appel

```
nomMethode(<liste de paramètres effectifs>)
```

- **<liste de paramètres effectifs>**

- Liste d'expressions séparées par des virgules et dont le nombre et le type correspond (compatible au sens de l'affectation) au nombre et au type des paramètres de la méthode

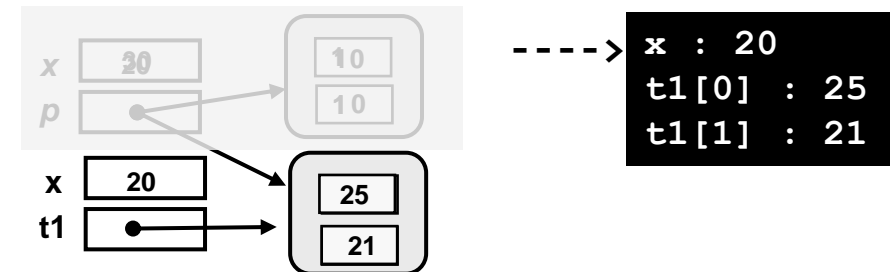
déclaration	appel (invocation)
<pre>static void afficher(){ ... }</pre>	<pre>afficher()</pre>
<pre>static double min(double a, double b){ ... }</pre>	<pre>min(10.5,x) min(x + y * 3,i)</pre>
<pre>static boolean contient(int[] tab,int val){ ... }</pre>	<pre>contient(tab1,14) contient(tab2,i)</pre>

Méthodes statiques - invocation

- 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 l'argument (le paramètre effectif).

```
Class Bidon {  
    ...  
    static void foo(int x, int[] p) {  
        ...  
        for (int i = 0; i < p.length; i++)  
            p[i] += 10;  
        x = x + 10;  
        p = new int[2];  
        p[0] = 10;  
        p[1] = 10;  
        ...  
    }  
}
```

```
int[] t1 = new int[2];  
t1[0] = 15;  
t1[1] = 11;  
int x = 20;  
foo(x,t1);  
System.out.println("x " + x);  
System.out.println("t1[0] " + t1[0]);  
System.out.println("t1[1] " + t1[1]);
```



Méthodes statiques - invocation

- Toutes les méthodes statiques d'une classe peuvent 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 m1() {  
        ...  
    }  
    static void m2() {  
        m1();  
        m3();  
    }  
    → static void m3() {  
        ...  
    }  
}
```

- Pour invoquer une méthode statique déclarée dans une autre classe il faut la préfixer par le nom de cette dernière
 - exemple : pour utiliser la méthode **random** définie dans la classe **Math**
- pour les classes définies dans un autre package que **java.lang** nécessité d'avoir une instruction **import**
 - exemple : pour utiliser la méthode **sort** définie dans la classe **Arrays** définie dans le package **java.util**

```
Math.random();
```

```
import java.util.Arrays;
```

à mettre avant la déclaration de la classe

```
Arrays.sort(monTableau);
```