

Support de cours

Cours:

## Introduction à la programmation orientée objet (en Java)

Vidéo:

### W14-03-final-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Classes final. Objet de type. Variable de type. Fait possible. Attribut final. Liberté du programmeur d'extensions. Constructeur de la classe. Comportement d'une classe. Méthode substring. Attributs d'une classe. Paramètres de méthodes. Petit programme principal. Chose de mauvais. Programmeur de la classe. Variable final.**



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en Java).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

## Le modificateur `final` (Partie 2)

### Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



Les méthodes et classes finales peuvent être à priori « agaçantes » :

- ▶ Exemple : la classe prédéfinie `String` est finale
- ▶ Aucune possibilité de définir

```
class MyString extends String
```

afin d'améliorer certaines méthodes par redéfinition !!

- ☞ Mais, permet de **fixer** une fois pour **toute le comportement d'une classe ou méthode**



Les méthodes des « classes final » sont a priori un peu agaçantes, en effet, elles brident la liberté du programmeur d'extensions qui souhaiterait étendre une hiérarchie existante ou redéfinir des méthodes héritées de plus haut. Par exemple, la classe prédéfinie « `String` » de l'API de Java est en fait déclarée comme « final », ce qui signifie que vous ne pourrez jamais écrire une classe « `MyString` » héritant de « `String` » et qui améliore éventuellement certaines méthodes par redéfinition ou qui en ajoute d'autres. Ceci à néanmoins pour avantage de fixer une fois pour toute le comportement d'une classe ou d'une méthode. Supposons en effet qu'il soit possible de faire ceci, c'est-à-dire possible de créer une sous-classe « `MyString` » de la classe prédéfinie « `String` ». Supposons que le programmeur de la classe « `MyString` » décide de redéfinir dans sa sous-classe des méthodes existantes dans la classe « `String` », par exemple, la méthode `substring` qui permet d'extraire une sous-chaîne

notes

résumé

0m 1s



Les méthodes et classes finales peuvent être à priori « agaçantes » :

- ▶ Exemple : la classe prédéfinie `String` est finale
- ▶ Aucune possibilité de définir

```
class MyString extends String
```

afin d'améliorer certaines méthodes par redéfinition !!

```
MyString
... substring(...)
{
}
```

- ☞ Mais, permet de **fixer** une fois pour **toute le comportement d'une classe ou méthode**

```
String s = new MyString();
s.substring(...) → /
```

de la chaîne de caractères. En fait, rien n'oblige ce programmeur, même si c'est tout à fait déconseillé de le faire, de respecter la sémantique connue de `substring`. Donc, il peut programmer une méthode `substring` qui aurait un comportement très différent de celle présente dans la classe « `String` ». C'est évidemment quelque chose à ne pas faire, quelque chose de mauvais, mais c'est tout à fait possible, il n'y a pas de garde-fous pour l'en empêcher. Il devient alors possible par exemple, de déclarer une variable de type « `String` » et de lui affecter un objet de type « `MyString` », puis d'appliquer la méthode `substring` à cet objet, puisque nous avons systématiquement la résolution dynamique des liens en Java, c'est la méthode `substring` de la classe « `MyString` » qui va s'appliquer ici donnant lieu à un comportement qui peut être inattendu.

notes

résumé

1m 1s



Si l'on ajoute `final` à une variable d'instance, une variable locale ou un paramètre :

- il devient impossible de lui affecter une valeur plus d'une fois

Un attribut `final` peut être initialisé dans le constructeur mais ne doit plus être modifié par la suite

**Attention :** `final` empêche l'affectation d'une nouvelle valeur à une variable, mais n'empêche pas de modifier l'éventuel objet référencé par cette variable :

Un exemple ...

```
class Personnage {  
    private final int  
        AGE_MAX;  
  
    public Personnage () {  
        AGE_MAX = 900;  
    }  
}
```

notes

résumé

2m 1s



Si l'on ajoute `final` à une variable d'instance, une variable locale ou un paramètre :

- il devient impossible de lui affecter une valeur plus d'une fois

Un attribut `final` peut être initialisé dans le constructeur mais ne doit plus être modifié par la suite

```
class Personnage {  
    private final int  
        AGE_MAX;  
  
    public Personnage () {  
        AGE_MAX = 900;  
        AGE_MAX =  
    }  
}
```

**Attention :** `final` empêche l'affectation d'une nouvelle valeur à une variable, mais n'empêche pas de modifier l'éventuel objet référencé par cette variable :

Un exemple ...

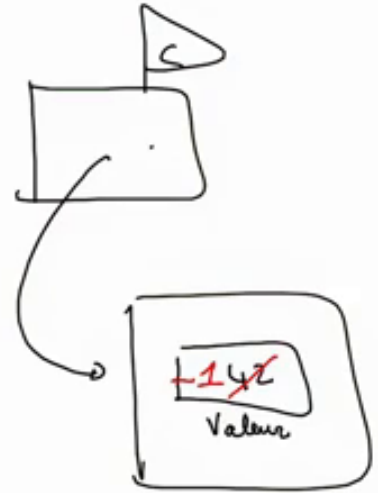
l'un ou l'autre. Donc par exemple, je peux choisir uniquement l'option avec le constructeur et dans ce cas-là le programme correct ressemblera à ceci. Si je retiens cette option, tout constructeur de la classe devra comporter une ligne d'initialisation de cet « attribut final », autrement il y aura un message d'erreur du compilateur indiquant que cette « variable final » n'a pas été initialisée. Une fois initialisé, cet « attribut final », comme pour une « variable final » quelconque ne peut plus être modifié, c'est-à-dire que ici par exemple si on imagine que dans le constructeur ici un peu plus loin on essaie de changer la valeur

notes

résumé

```
class Conteneur {
    private int valeur;
    public void setValeur(int val) { valeur = val; }
}

class Test {
    public static void main(String[] args) {
        Conteneur c = new Conteneur();
        → c.setValeur(42);
        modifier(c);
    }
    static void modifier(final Conteneur c) {
        c.setValeur(-1); // modifie l'objet référencé !!
        //c = new Conteneur(); //FAUX
    }
}
```



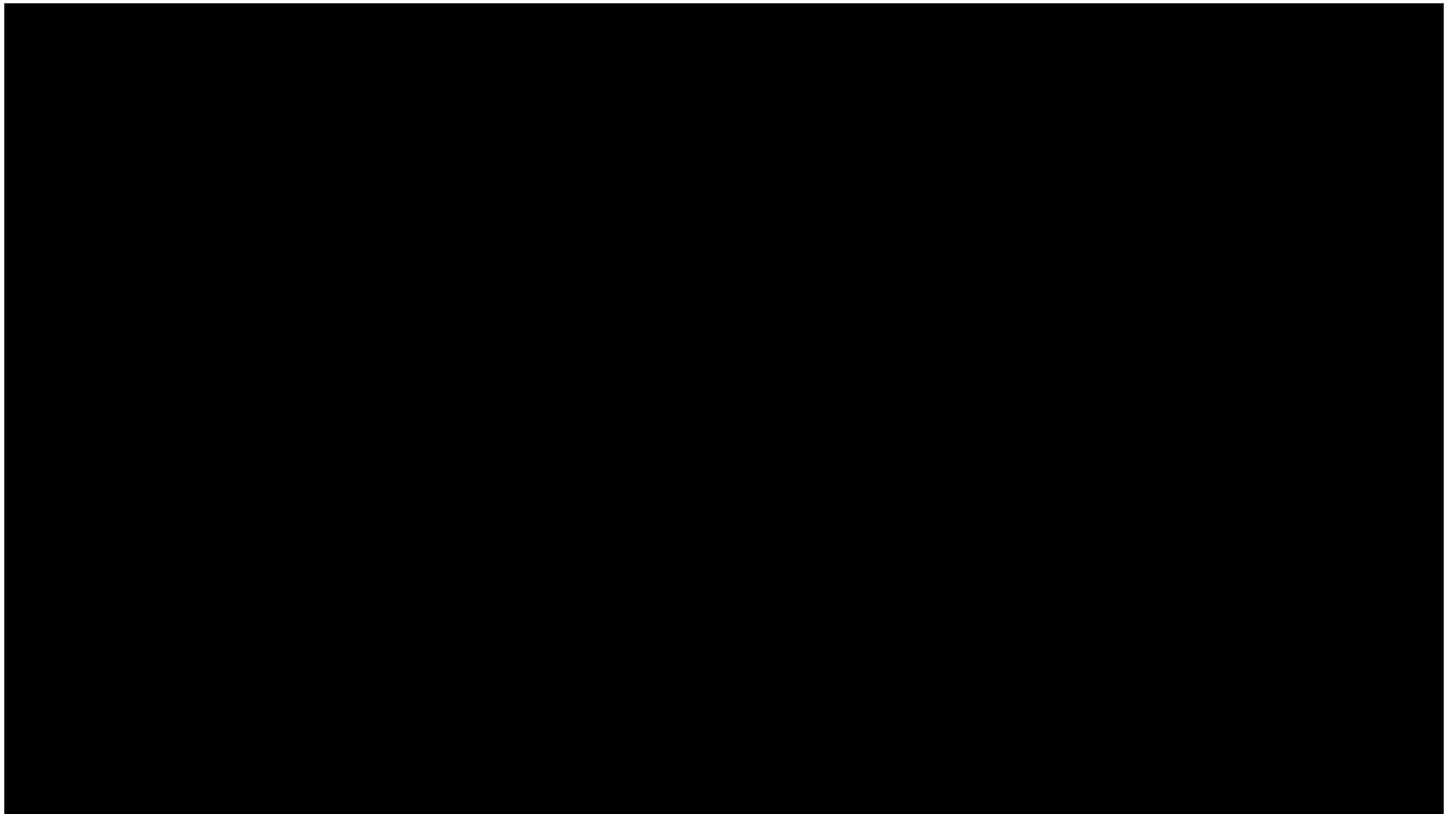
de cet attribut, et bien on aura un message d'erreur du compilateur. Mais attention, il faut bien comprendre que lorsqu'une « variable final » contient la référence vers un objet, alors on ne peut pas affecter une nouvelle valeur à cette variable certes, mais rien n'empêche de modifier l'objet référencé via cette variable. Examinons ce point sur un exemple. Donc, nous avons ici le cas d'une classe « Conteneur » qui contient un attribut entier qui s'appelle valeur. La classe « Conteneur » dispose uniquement d'une méthode public setValeur qui permet de modifier la valeur de l'attribut. Nous avons ensuite un petit programme principal qui déclare un objet de type « Conteneur » et qui affecte une valeur à son attribut entier. La situation en mémoire après l'exécution de cette ligne de code sera donc la suivante ; nous avons dans la variable c une référence vers un objet de type « Conteneur », dont l'attribut valeur vaut 42. Le programme principal appelle une méthode modifier qui prend c en paramètre. La méthode modifier a pour particularité que le conteneur passé en paramètre est déclaré comme « final ». On pourrait donc imaginer que « final » signifie que l'objet passé en paramètre ne soit pas modifiable dans la méthode modifier. Or, ça dépend ce qu'on entend par modifiable, certes on ne peut pas modifier l'argument lui-même en lui affectant une nouvelle référence, par contre l'objet référencé est tout à fait modifiable via c, même si c est passé comme « paramètre final », ce qui signifie qu'au terme de l'exécution de cette ligne, et bien, l'objet référencé aura bel et bien été modifié. En clair, le fait que le paramètre soit « final » empêche

## notes

## résumé

5m 1s





que la variable passée en argument pointe vers un autre objet, donc ceci n'est pas possible, il n'est pas possible de modifier la référence elle-même, par contre rien n'empêche de modifier l'objet référencé via ce « paramètre final ». Ceci est évidemment valable pour toute « variable final » contenant la référence vers un objet, que se soit un attribut, une variable locale ou un paramètre de méthode. Donc concrètement, lorsqu'on affecte un objet à une « variable final », cela ne signifie pas que l'objet est préservé de toute modification par l'extérieur, cela signifie simplement que la variable ne peut référencer que cet objet. Enfin, un petit détail auquel vous aurez peut-être été attentif, puisque Java n'utilise que le passage par valeur et que donc tout paramètre ne peut être modifié à l'intérieur de la méthode tout en préservant cette modification à l'extérieur de la méthode, à quoi cela peut-il bien servir de déclarer ce paramètre comme « final » ? de déclarer ce paramètre comme « final » ?

#### notes

---

---

---

---

---

---

---

---

---

---

#### résumé

7m 1s



---

---

---

---

---

---

---

---

---

---