

Support de cours

Cours:

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

Vidéo:

### W13-03-heritconstr-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Classe rectangle. Classe figuregeometrique. Moyen d'un constructeur. Constructeurs de la classe rectangle. Instanciation d'une sous-classe. Constructeur de rectangle. Instance r de la classe rectangle. Constructeurs. Constructeur explicite. Séquence vidéo. Constructeur de la classe figuregeometrique. Ensemble des attributs de la classe. Attribut position. Constructeur des sous-classes. Concepteur de la classe rectangle.**



[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/>

# Héritage : constructeurs

## (Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



# Constructeurs et héritage

FigureGeometrie  
↑  
Rectangle EPFL

Lors de l'instanciation d'une sous-classe, il faut initialiser :

- ▶ les attributs *propres à la sous-classe*
- ▶ les attributs *hérités des super-classes*

```
Rectangle r = new Rectangle (2.5, 3.4);  
Rectangle (double h, double l) {...}
```

**MAIS...**

...il ne doit pas être à la charge du concepteur des sous-classes de réaliser lui-même l'*initialisation des attributs hérités*

L'accès à ces attributs pourrait notamment être interdit ! (**privat**)

L'initialisation des attributs hérités doit donc se faire au niveau des classes où ils sont explicitement définis.

**Solution** : l'initialisation des attributs hérités doit se faire en **invquant les constructeurs des super-classes**.

Dans cette séquence vidéo, nous allons nous intéresser aux conséquences de l'héritage sur les constructeurs, c'est-à-dire l'initialisation. Nous avons vu dans une séquence vidéo précédente sur les constructeurs que lors d'une instanciation d'une sous-classe, il nous fallait initialiser les attributs. C'est ce que nous faisons, par exemple, quand on déclare une instance r de la classe Rectangle en passant par exemple la largeur et la hauteur. Ceci était fait au moyen d'un constructeur qui était en charge d'initialiser des attributs. Mais si la classe Rectangle hérite d'une classe FigureGeometrique, alors la classe Rectangle reçoit l'ensemble des attributs de la classe FigureGeometrique. Donc les constructeurs de la classe Rectangle ont pour charge d'initialiser les attributs de la classe Rectangle, y compris les attributs hérités de la super-classe FigureGeometrique. Cependant, ce ne doit pas être le concepteur de la classe Rectangle qui doit lui-même initialiser les attributs de la classe FigureGeometrique, il serait peut-être même bien incapable de le faire

notes

résumé

0m 1s



## Constructeurs et héritage : appel explicite

L'invocation du constructeur de la super-classe se fait au tout début du corps du constructeur au moyen du mot réservé `super` .

### Syntaxe :

```
SousClasse(liste de paramètres)
{
    /* Arguments : liste d'arguments attendus par
     * un des constructeurs de la super-classe de SousClasse
     */
    super(Arguments);
    // initialisation des attributs de SousClasse ici
}
```

Lorsque la super-classe admet un constructeur par défaut, l'invocation explicite de ce constructeur dans la sous-classe n'est pas obligatoire

☞ le compilateur se charge de réaliser l'invocation du constructeur par défaut

si la super-classe avait des attributs privés ; ils ne pourraient pas y accéder. Donc, comment le concepteur de la sous-classe Rectangle peut-il initialiser les attributs de la super-classe FigureGeometrique ? Pour cela, il doit faire appel dans le constructeur de Rectangle au constructeur de la classe FigureGeometrique. L'initialisation des attributs hérités doit se faire en appelant les constructeurs des super-classes dans le constructeur des sous-classes. Voyons comment cela se fait en Java. Pour pouvoir appeler le constructeur de la super-classe dans le constructeur de la sous-classe, il suffit d'utiliser la méthode `super`. Et pour ça, il faut bien sûr que cet appel soit fait au tout début du corps de constructeur de la sous-classe. Donc la syntaxe sera la suivante : dans le constructeur de la sous-classe, qui peut recevoir différents paramètres, tout au début de son corps, on aura ici un appel à un des constructeurs de la super-classe en écrivant simplement comme cela le mot-clé `super` et puis en passant à ce constructeur les arguments nécessaires pour le constructeur de la super-classe que l'on veut appeler. Et puis ensuite, derrière cette première ligne du constructeur de la sous-classe, on pourra continuer à initialiser des attributs de la sous-classe. A noter que si la super-classe admet un constructeur par défaut, il n'est pas nécessaire d'écrire explicitement l'appel au constructeur de la super-classe,

### notes

### résumé

1m 13s



## Constructeurs et héritage : exemple 1

Si la classe parente n'admet pas de constructeur par défaut, l'**invocation explicite** d'un de ses constructeurs **est obligatoire** dans les constructeurs de la sous-classe

☞ La sous-classe doit admettre *au moins un constructeur explicite*.

Exemple :

```
class FigureGeometrique {
    private Position position;
    public FigureGeometrique(double x, double y) {
        position = new Position(x,y);
    }
    // ...
}

class Rectangle extends FigureGeometrique {
    private double largeur;
    private double hauteur;
    public Rectangle(double x, double y, double l, double h) {
        super(x,y);
        largeur = l; hauteur = h;
    } // ...
}
```

le compilateur se charge pour vous d'appeler le constructeur par défaut de la super-classe. Regardons en détails un exemple. Tout d'abord, si la classe n'a pas de constructeur par défaut, alors à ce moment-là, il faut absolument écrire explicitement l'appel à un des constructeurs de la super-classe. Donc évidemment, cela suppose que la sous-classe a aussi elle-même un constructeur explicite où on peut faire l'écriture de cet appel au constructeur de la super-classe. Par exemple, si l'on considère la classe FigureGeometrique avec un constructeur ici explicite et pas de constructeur par défaut, un constructeur explicite qui prend deux paramètres pour pouvoir initialiser un attribut position, ici, donc on supposerait qu'il a un constructeur comme ceci a deux paramètres x, y pour fixer la position de cette figure géométrique. Puis on a une classe Rectangle, ici, qui hérite de la classe FigureGeometrique, et dans cette classe Rectangle, on va avoir un constructeur qui va recevoir les paramètres pour pouvoir initialiser la position de sa super-classe FigureGeometrique ; un Rectangle est une FigureGeometrique et donc récupère par héritage la position de la figure géométrique, donc un rectangle aura aussi une position qu'il va falloir initialiser, et puis bien sûr, comme d'habitude, les paramètres l et h pour pouvoir initialiser ses propres attributs largeur et hauteur. Donc ce constructeur, ici, qui prend ces quatre paramètres,

notes

résumé

2m 37s



Autre exemple (qui ne fait pas la même chose) :

```
class FigureGeometrique {  
    private Position position;  
    public FigureGeometrique() { position = new Position(0.0, 0.0); }  
}  
class Rectangle extends FigureGeometrique {  
    private double largeur;  
    private double hauteur;  
    public Rectangle(double l, double h) {  
        largeur = l;  
        hauteur = h;  
    }  
    // ...  
}
```

va commencer par appeler le constructeur de la super-classe en écrivant le mot-clé `super`, puis en passant de deux paramètres, le constructeur de la super-classe qui prend les deux paramètres `x` et `y`. Donc ici on va avoir un appel à ce constructeur, lui-même initialisera donc la position. Et puis ensuite, comme d'habitude, on continue en initialisant les attributs propres `largeur` et `hauteur` avec les paramètres reçus `l` et `h`. Dans le cas où la super-classe `FigureGeometrique` a un constructeur par défaut, alors il n'est plus nécessaire de faire un appel explicite au constructeur de la super-classe dans le constructeur de la sous-classe, donc ici, par exemple, on a notre classe `FigureGeometrique` qui a donc un constructeur par défaut, c'est-à-dire un constructeur qui ne prend pas de paramètres, qui initialiserait par exemple la position en 0.0 (zéro zéro). Donc dans ce cas-là, la sous-classe `Rectangle` n'a pas besoin de faire un appel explicite au constructeur par défaut et on l'a muni donc de son constructeur usuel ici, qui prend les deux paramètres `l` et `h` pour initialiser sa largeur et sa hauteur. mais on voit qu'ici il n'y a pas d'appel explicite au constructeur de la super-classe, en fait il y a, de fait, un appel au constructeur par défaut de la super-classe qui est fait automatiquement par le compilateur. par le compilateur.

notes

résumé

4m 1s

