

Support de cours

Cours:

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

Vidéo:

### W17-05-copieprofonde-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Copie polymorphique. Constructeur de la classe. Constructeurs de copies. Méthode de copie polymorphique. Type de l'instance réelle. Constructeur de copie. Copie profonde. Constructeurs de copies de la superclasse. Copie de l'objet. Constructeurs. Point. Recours. Méthode. Idée. Mécanisme.**



[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/>

## Etude de cas : copie profonde (Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



```
public Montre(Montre autre)
{
    super(autre);
    coeur = new Mecanisme(autre.coeur);
    accessoires = new ArrayList<Accessoire>();
    for (Accessoire acc : autre.accessoires) {
        accessoires.add(new Accessoire(acc));
    }
}
```

non polymorphique!

Mecanisme Double

La réponse est non. Car, comme nous l'avons vu dans le cours, les constructeurs sont par essence non polymorphiques. Le constructeur de la classe « Mecanisme » ne pourra donc pas s'adapter

notes

résumé

0m 1s



```
public Montre(Montre autre)
{
    super(autre);
    coeur = new Mecanisme(autre.coeur);
    accessoires = new ArrayList<Accessoire>();
    for (Accessoire acc : autre.accessoires) {
        accessoires.add(new Accessoire(acc));
    }
}
```

non polymorphique!

Mecanisme Double

au type de l'instance réelle qu'il essaie de copier. Il ne va donc ici voir l'objet copié que comme un « Mecanisme », et ne va donc en copier que sa partie « Mecanisme ». Donc toutes les spécificités de « MecanismeDouble » vont disparaître.

notes

résumé

0m 25s



```
public Montre(Montre autre)
{
    super(autre);
    coeur = new Mecanisme(autre.coeur);
    accessoires = new ArrayList<Accessoire>();
    for (Accessoire acc : autre.accessoires) {
        accessoires.add(new Accessoire(acc));
    }
}
```

Mais... comment copier chaque élément en tant que tel ?  
(et non pas comme une instance des super-classes `Mecanisme` et `Accessoire`)

☛ **copie polymorphique**

Plus de date et plus de réveil, donc.

notes

résumé

0m 37s



```
public Montre(Montre autre) {  
    super(autre);  
    coeur = autre.coeur.copie();  
    accessoires = new ArrayList<Accessoire>();  
    for (Accessoire acc : autre.accessoires) {  
        accessoires.add(acc.copie());  
    }  
}
```

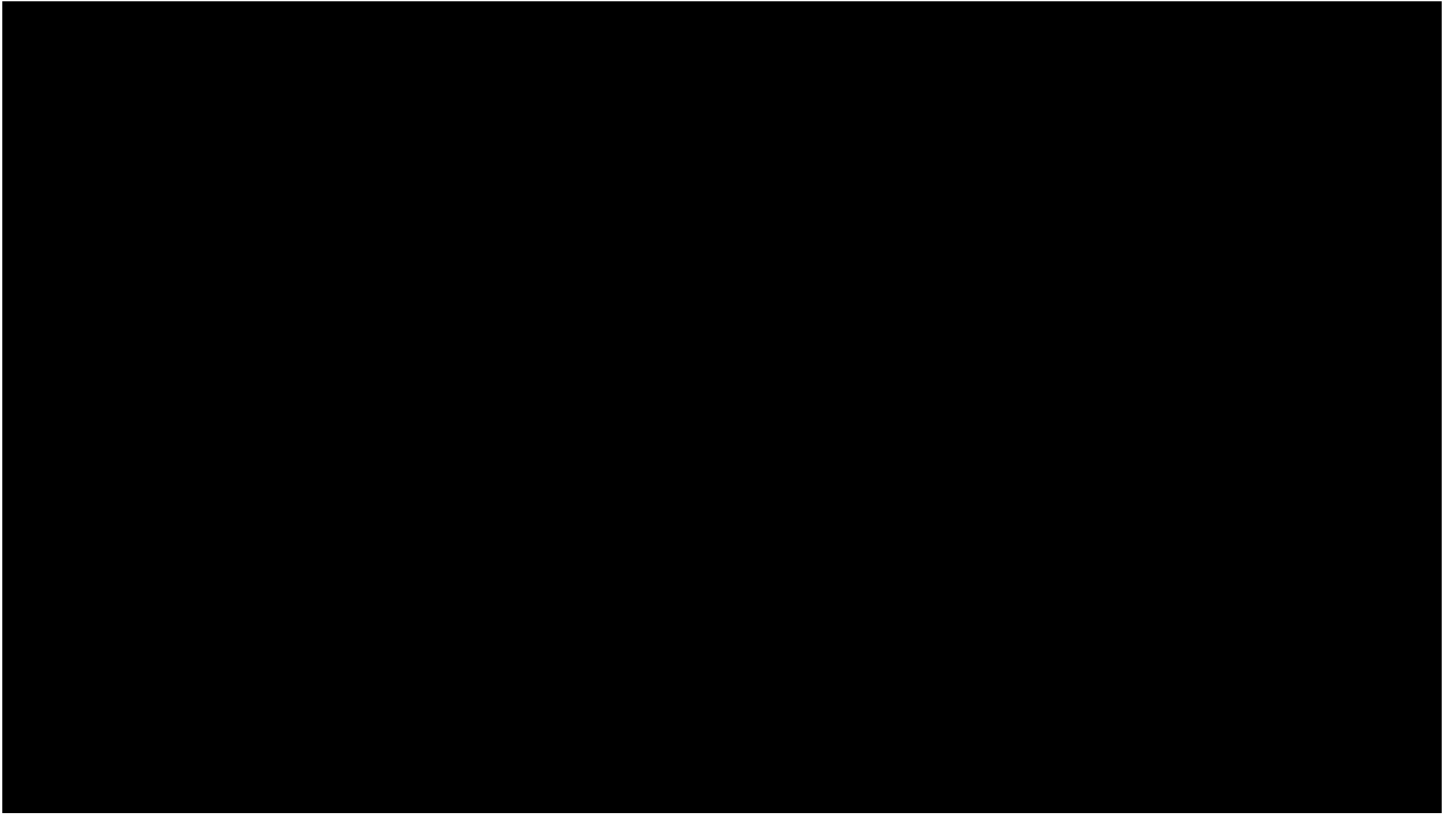
La question qui se pose est donc : comment copier chaque élément, que ce soit un « Accessoire » ou un « Mekanisme », en tant que tel ? C'est-à-dire que si c'est un « MekanismeDouble », il sera copié comme un « MekanismeDouble ». Si c'est un « Bracelet », il sera copié comme un « Bracelet »; Donc, comment réaliser de la copie polymorphique ? Pour réaliser une copie polymorphique, on ne peut pas avoir recours aux constructeurs de copies, nous venons de le voir, il suffit d'avoir recours à une méthode pouvant être, elle, polymorphique. L'idée serait donc de définir une méthode de copie polymorphique permettant de copier des « Mekanismes » et une méthode de copie polymorphique qui permettrait de copier des accessoires. Nous allons voir comment écrire précisément une méthode de copie polymorphique, pour les « Accessoires » par exemple. La même chose pourra être appliquée pour les « Mekanismes », et cette façon de procéder va nous permettre réellement d'atteindre la copie profonde,

notes

résumé

0m 42s





celle qui permet d'obtenir une copie de l'objet copié, qui est totalement indépendante de cet objet. Enfin, un point que je n'ai pas commenté jusqu'ici mais qui a son importance, lorsqu'on écrit un constructeur de copie, il est impératif de ne pas oublier de faire appel aux constructeurs de copies de la superclasse. Sinon, si l'on oublie cet appel, que se passe-t-il ici ? que se passe-t-il ici ?

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

1m 37s



.....

.....

.....

.....

.....