

Support de cours

Cours:

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

Vidéo:

### W17-02-affichage-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Exemple de type montre. Variable de type produit. Méthode d'affichage. Façon polymorphique. Hiérarchie de produits. Méthode toString. Valeur de base. Différents produits. Étape précédente. Façon propre. Objet de type montre. Séquence précédente. Attribut de type double. Méthode toString de façon. Calcul du prix du produit.**



[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 : affichage polymorphique

## (Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

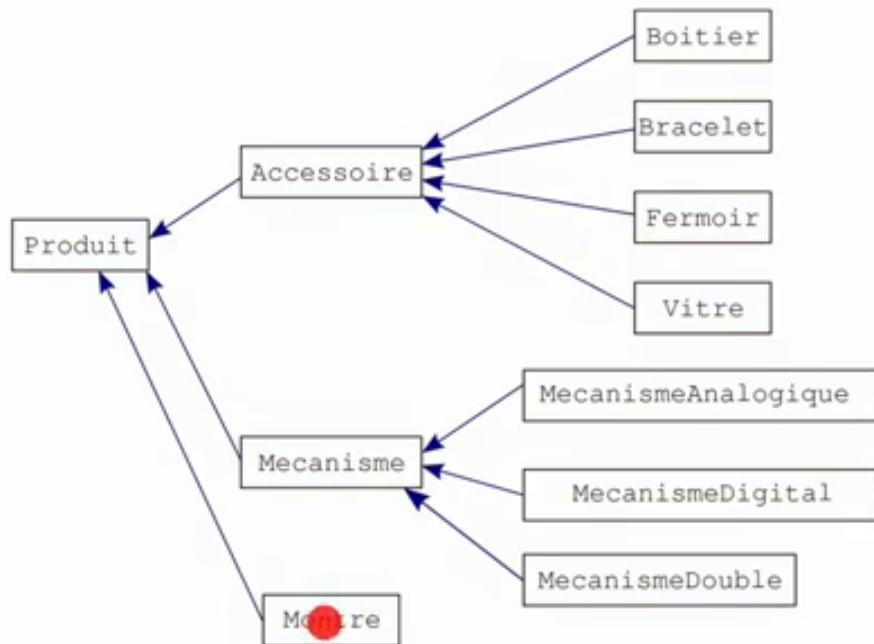
...

notes

résumé

0m 0s





Dans cette séquence, nous poursuivons notre étude de cas sur les montres, suisses ou pas d'ailleurs, et nous allons aborder l'affichage polymorphique des différents produits de notre hiérarchie. Pour rappel donc, à l'étape précédente nous avons ébauché une hiérarchie de produits et nous nous intéressons maintenant à comment faire en sorte que ces différents produits puissent s'afficher de façon polymorphique. Concrètement, un produit comme une montre, un mécanisme ou un accessoire aura sa façon propre d'être affiché et nous voulons que cet affichage soit polymorphique, c'est-à-dire que, si nous mettons un objet, par exemple

notes

résumé

0m 1s



- Tous les produits sont « affichables », chacun à sa façon

☛ affichage polymorphique ?

```
Produit p = new Montre (...);  
System.out.println(p);
```

de type Montre, dans une variable de type Produit et que nous invoquons la méthode d'affichage sur la variable en question, eh bien l'affichage s'adapte automatiquement au type réel de l'instance stocké dans la variable. Chaque produit a donc sa propre façon de s'afficher et nous nous posons la question de comment procéder pour faire un affichage polymorphique. Alors, l'idée serait par exemple que je déclare une variable de type produit, que je lui affecte la référence à un objet de type Montre, une sous-classe donc de Produit, par exemple, et qu'ensuite je procède à l'affichage de cet objet "p" et j'aimerais donc avoir toutes les caractéristiques de l'objet

notes

résumé

0m 37s



- Tous les produits sont « affichables », chacun à sa façon

☞ affichage polymorphique ?

Le plus naturel est de redéfinir la méthode `toString` héritée de `Object` :

```
class Produit {  
    @Override  
    public String toString() {  
        //...  
    }  
}  
// -----
```

en question, mais qui s'adaptent automatiquement au fait que l'objet est en réalité une montre, c'est-à-dire que ce que j'ai stocké dans la variable "p" est une Montre.

notes

résumé

1m 25s



Précisons son comportement par défaut :

```
@Override
public String toString() {
    return Double.toString(valeur);
}
```

Le plus naturel est évidemment d'avoir recours à la méthode `toString`, qui est invoquée automatiquement par `system.out.println`, et si l'on veut que chaque produit soit affichable à sa façon, il suffit de redéfinir cette méthode `toString` de façon appropriée dans toutes les sous-classes où on estime nécessaire de le faire.

notes

résumé

1m 32s



## Affichage par défaut

```
class _Product { EPFL  
    public  
    private double valeur;  
}
```

Précisons son comportement par défaut :

```
@Override  
public String toString() {  
    return Double.toString(valeur);  
}
```

Supposons que nous souhaitons faire en sorte que l'affichage par défaut d'un produit en affiche le prix. Pour rappel, dans la séquence précédente, nous avons décrit un produit comme étant caractérisé par un attribut de type double qui correspondait à sa valeur de base, et la classe produit contenait également une méthode publique

notes

résumé

1m 49s



## Affichage par défaut

```
class _Produit { EPFL
    public double prix() {
        return valeur;
    }
    private double valeur;
}
```

Précisons son comportement par défaut :

```
@Override
public String toString() {
    return Double.toString(valeur);
}
```

permettant le calcul du prix du produit et qui, dans la classe Produit, retournerait simplement la valeur de base. Dans la classe Produit, pour faire en sorte que la méthode toString affiche une représentation sous forme de chaîne de caractères du prix du produit,

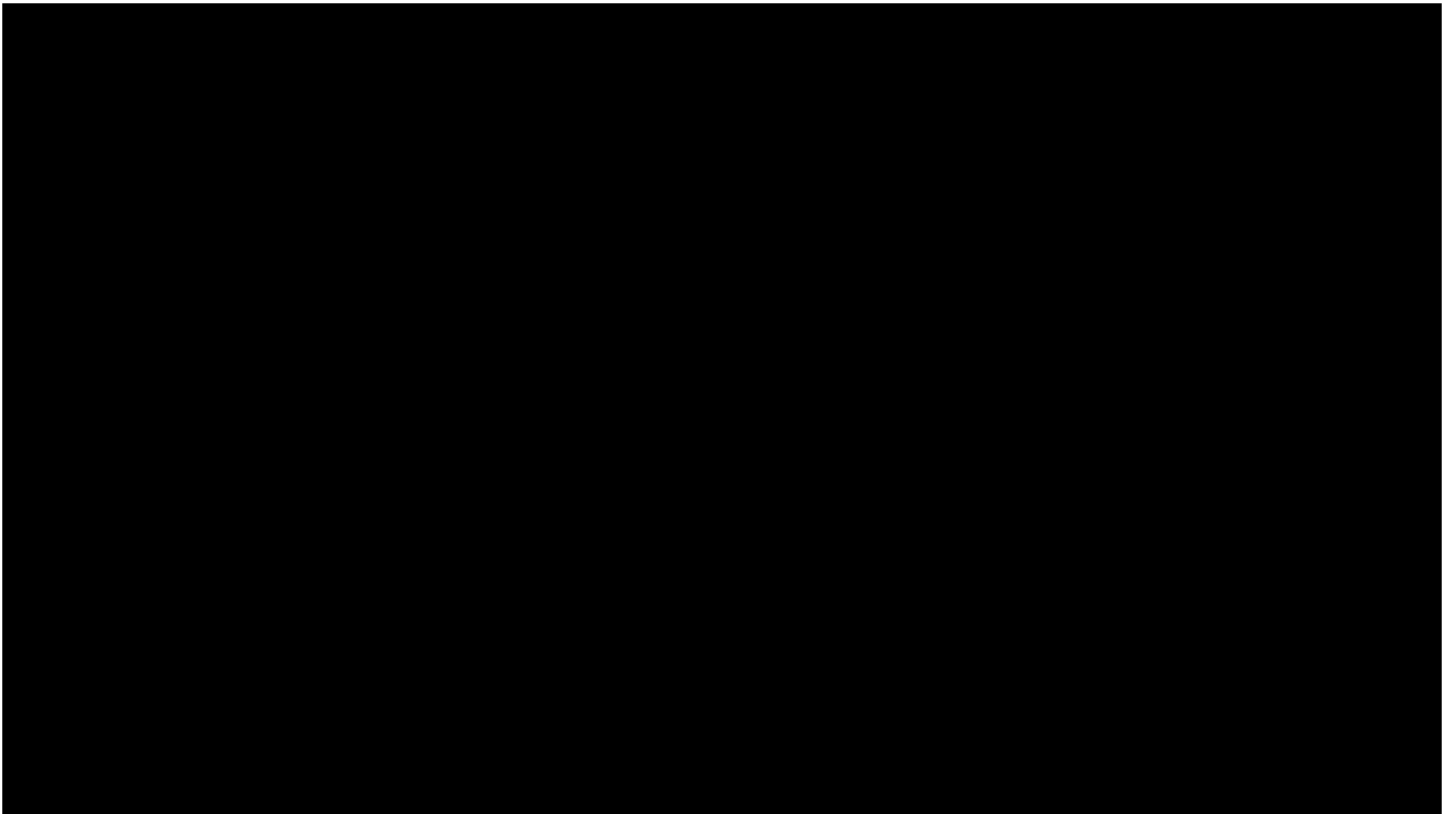
notes

résumé

2m 13s







on pourrait donc être tenté de faire afficher le prix en utilisant directement la valeur de l'attribut, puisqu'on se dit que finalement, le prix n'est autre que la valeur de base du produit. Cette façon de faire n'est cependant pas bonne. Sauriez-vous dire pourquoi ? Sauriez-vous dire pourquoi ?

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

2m 37s



.....

.....

.....

.....

.....