

Support de cours

Cours:

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

Vidéo:

### W17-02-affichage-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Valeur de base. Prix correct du produit. Calcul du prix d'un objet. Méthode toString telle. Sous-types de produits. Prix d'une montre. Premières choses. Objet de type montre. Produit de base. Façon correcte. Valeur de base quelconque. Prix d'un bracelet. Place de valeur. Classe produit. Attribut d'un type objet.**



[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 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

## Affichage par défaut

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

Précisons son comportement par défaut :

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

Nous voulons en effet que la méthode `toString` telle que définie dans la classe `Produit` détermine le comportement par défaut, c'est-à-dire qu'elle soit capable d'afficher le prix, le prix correct du produit, même si elle n'est pas redéfinie dans les sous-classes. Or, le calcul du prix d'un objet correspond à sa valeur de base pour un produit de base, mais ce n'est pas nécessairement le même calcul que l'on effectuerait pour des sous-types de produits. Par exemple, le prix d'un bracelet pourrait correspondre à une valeur de base; par contre, le prix d'une montre pourrait être calculé comme étant la somme du prix de tous ses composants.

notes

résumé

0m 1s



Précisons son comportement par défaut :

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

Et dans ce cas, si l'on applique la méthode `toString` à un objet de type `Montre`, on veut que ce soit le prix de la montre qui s'affiche correctement, et non pas une valeur de base quelconque. La façon correcte de procéder ici est donc d'utiliser à la place de valeur la méthode `prix`, laquelle pourra bien sûr elle-même avoir un comportement polymorphique. Enfin, vous aurez noté au passage qu'ici il a été nécessaire

notes

résumé

0m 37s



Supposons enfin que :

- ▶ un produit n'existe pas en tant que tel : c'est une *abstraction*
- ▶ l'on fixe la valeur de base d'un produit au départ et que l'on ne puisse pas en changer

```
abstract class Produit {  
  
    private final double valeur;  
  
    public Produit(double uneValeur) {  
        valeur = uneValeur;  
    }  
}
```

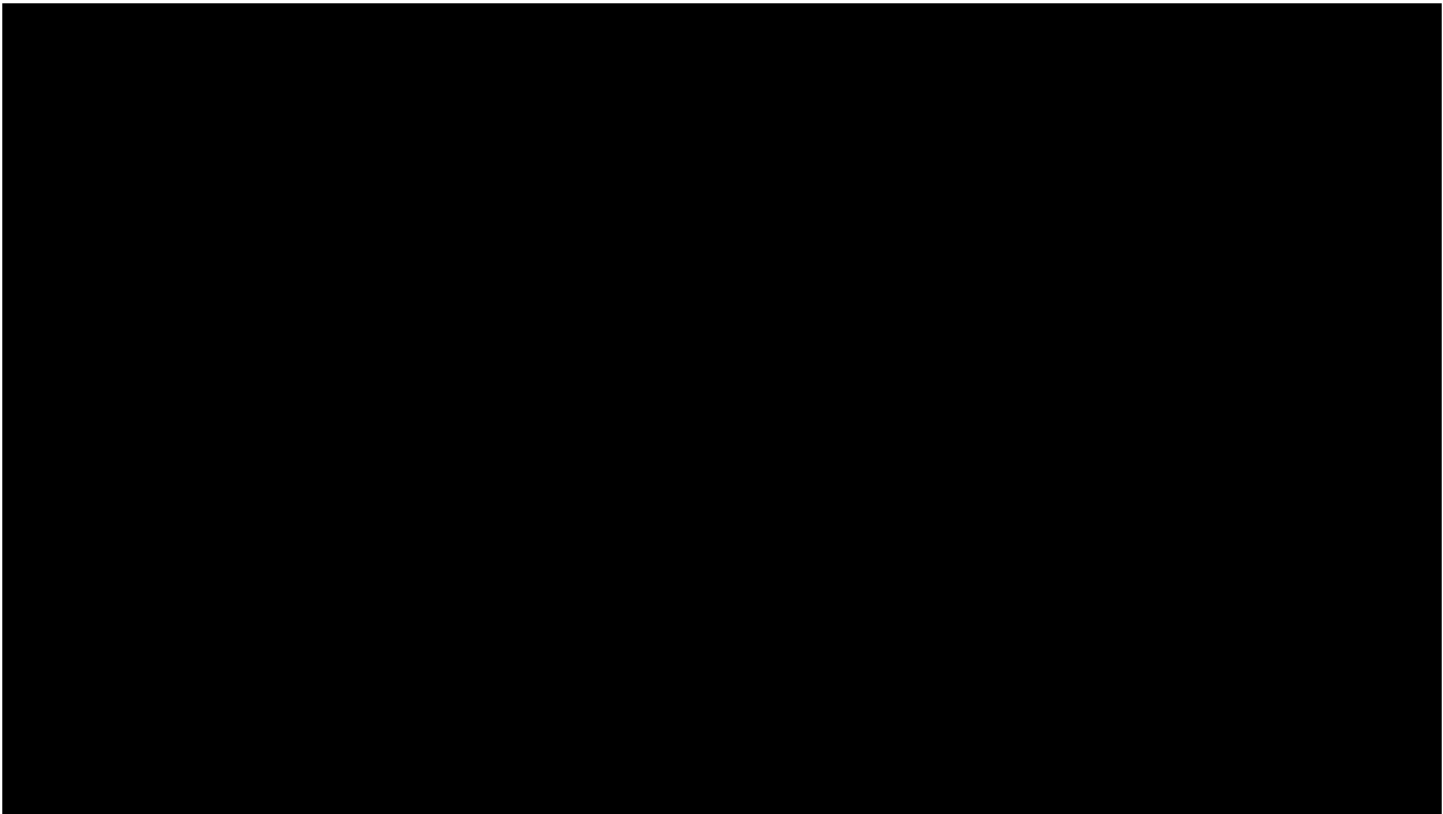
de convertir le double retourné par prix en une string au moyen de cette tournure. Finalisons maintenant notre classe Produit. Une des premières choses à laquelle il est naturel de penser, est qu'un produit n'a pas d'existence en tant que tel. Il s'agit d'une entité abstraite dans notre conception. Nous pouvons rendre ceci clair, au niveau de la conception, en étiquetant la classe Produit comme étant abstraite. Supposons maintenant que l'on souhaite imposer le fait que la valeur d'un produit, une fois initialisée, ne puisse plus changer de valeur en cours d'existence du produit. Donc, l'idée serait de pouvoir donner une valeur initiale à cet attribut, mais ensuite de ne pas permettre sa modification.

notes

résumé

1m 1s





Ceci est possible en étiquetant l'attribut en question comme étant final. Si valeur avait été un attribut d'un type objet, donc autre chose qu'un type de base, est-ce que le fait de l'étiqueter en final empêche toute modification de cet attribut ? de cet attribut ?

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

1m 37s



.....

.....

.....

.....