

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W17-04-interface-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Modélisation des différents mécanismes. Œuvre de constructeurs. Classe mecanismedigital. Classe mecanismeanalogue. Mécanisme analogique. Attribut spécifique. Mécanisme double. Mécanisme digital. Classe analogique. Description de ce constructeur. Classe mecanismedouble. Valeur de base du produit. Modélisation des mécanismes doubles. Sous-classes directes. Fois des caractéristiques des mécanismes.



[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 : modélisation des mécanismes

(Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Nous poursuivons notre étude de cas sur les montres

notes

résumé

0m 1s



- ▶ Les montres ont un *mécanisme* de base [...]
- ▶ Les *mécanismes* [...] sont aussi des produits
- ▶ Il existe trois sortes de *mécanismes* : *analogiques*, *digitaux* et *doubles*.
- ▶ Pour les *mécanismes doubles*, on supposera ici qu'ils n'indiquent qu'une seule heure, mais se comportent sinon à la fois comme des *mécanismes analogiques* et comme des *mécanismes digitaux*

```
class Mecanisme extends Produit {
}

class MecanismeAnalogique extends Mecanisme {
    date;
}

class MecanismeDigital extends Mecanisme {
    reveil;
}

class MecanismeDouble extends Mecanisme {
    date;
    reveil;
}

// =====
class Montre extends Produit {
    // ...
    private Mecanisme coeur;
    // ...
}
```

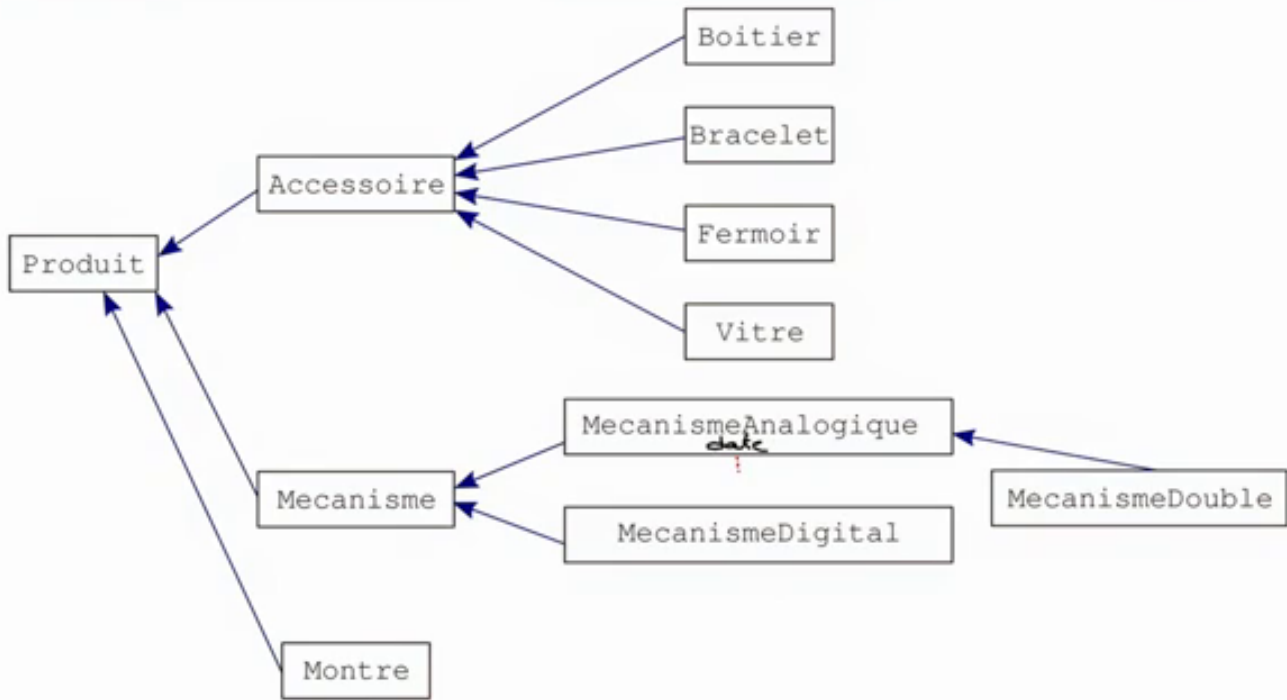
il s'agit d'affiner la modélisation des différents mécanismes appliqués en incorporant notamment la notion d'interface. Pour rappel, dans les séquences précédentes, le cœur d'une montre était modélisé comme étant un mécanisme un mécanisme pouvant se décliner sous différentes formes on pouvait avoir soit : un mécanisme analogique, un mécanisme digital ou encore un mécanisme double. Ces trois types de mécanismes héritaient tous de Mecanisme. Cette première hiérarchie n'est cependant pas tout à fait conforme à nos souhaits concernant la modélisation des mécanismes doubles. Nous souhaitons, en effet, qu'un mécanisme double puisse avoir à la fois des caractéristiques des mécanismes analogiques et des caractéristiques des mécanismes digitaux. Idéalement, un mécanisme double devrait pouvoir ici hériter de la classe MecanismeDigital ainsi que de la classe MecanismeAnalogique, Malheureusement, l'héritage multiple n'existe pas en Java. Supposons par exemple que dans la classe analogique nous ayons un attribut spécifique aux mécanismes analogiques qui serait la date affichée par le mécanisme et que dans le cas du mécanisme digital nous ayons un autre attribut qui serait un réveil assorti au mécanisme digital, alors maintenir ce lien d'héritage impliquerait une duplication des deux attributs dans le mécanisme double puisqu'on voudrait effectivement qu'un mécanisme double ait les caractéristiques des deux classes en question c'est-à-dire dispose d'un réveil et dispose d'une date. Nous nous trouverions donc ici dans cette situation et il en serait de même pour toutes les méthodes spécifiques aux mécanismes analogiques et aux mécanismes digitaux,

notes

résumé

0m 6s





qu'il faudrait reprendre dans leur intégralité dans les mécanismes doubles. On peut donc prendre le parti de considérer que, par exemple, un mécanisme double est un mécanisme analogique auquel se grefferaient des caractéristiques du mécanisme digital. Si l'on prend ce point de vue, on permet à l'héritage d'alléger les duplications induites par la conception initiale. Nous aboutissons donc à ce stade à une hiérarchie qui ressemble à ceci : les mécanismes doubles héritent des mécanismes analogiques, Il n'existe pas pour l'instant de lien direct entre les mécanismes doubles et les mécanismes digitaux, nous allons y remédier un peu plus tard. Tous les attributs spécifiques aux mécanismes analogiques et toutes les méthodes sont donc hérités par la classe MecanismeDouble qui n'a pas besoin de les dupliquer. En revanche, comme on ne peut pas hériter

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

1m 49s



```
class Mecanisme extends Produit {
    private String heure;
}
class MecanismeAnalogique extends Mecanisme {
    private int date;
}

class MecanismeDigital extends Mecanisme {
    private String reveil;
}

// Un mécanisme double est avant tout un mécanisme
// analogique auquel on ajoute des caractéristiques
// d'un mécanisme digital
class MecanismeDouble extends MecanismeAnalogique
{
    // duplication incontournable ici
    private String reveil;
}
```

de la classe MecanismeDigital, les caractéristiques du mécanisme digital que l'on voudrait voir reproduites dans les mécanismes doubles doivent être dupliquées. Le code Java qui correspond à cette première révision de la hiérarchie est donc le suivant. Nous avons donc une superclasse Mecanisme qui dérive de Produit, qui dispose de deux sous-classes directes : la classe MecanismeAnalogique et la classe MecanismeDigital ; on suppose donc ici que nos classes MecanismeAnalogique et MecanismeDigital ont des attributs spécifiques : une date pour les mécanismes analogiques un réveil pour les mécanismes digitaux. Et nous prenons donc le parti de considérer qu'un mécanisme double est avant tout un mécanisme analogique auquel on ajoute des caractéristiques du mécanisme digital. Donc ici le lien est un, le lien d'héritage est établi entre mécanisme double et mécanisme analogique, et nous greffons à la classe MecanismeDouble des éléments qui sont caractéristiques du mécanisme digital. Donc ici une duplication incontournable de l'attribut reveil.

notes

résumé

2m 37s



Pour le moment, il n'y a qu'un constructeur par défaut, mais fixons la construction des `Mecanisme` :

- ▶ initialisation de la valeur de base (`Produit`)
- ▶ initialisation de l'heure

```
class Mecanisme extends Produit {  
    private String heure;  
  
    public Mecanisme(double valeurDeBase, String uneHeure) {  
        super(valeurDeBase);  
        heure = uneHeure;  
    }  
    public Mecanisme(double valeurDeBase){  
        super(valeurDeBase);  
        heure = "12:00";  
    }  
}
```

Intéressons nous maintenant à la construction des mécanismes : la mise en œuvre de constructeurs dans une hiérarchie de classe obéit à certaines règles que nous allons revoir maintenant. Commençons donc par fixer la construction des mécanismes - jusqu'ici nous n'avions qu'un constructeur par défaut - affinons un petit peu la description de ce constructeur. Sachant qu'un mécanisme est un produit, donc hérite de la classe `Produit`, il doit initialiser la valeur de base du produit hérité de `Produit` et doit initialiser son attribut propre, à savoir l'heure. On peut donc naturellement pour la classe `Mecanisme` penser à un constructeur qui aurait cette allure donc il prendrait en paramètre une valeur permettant d'initialiser l'attribut hérité de `Produit` et prendrait en paramètre une seconde valeur permettant d'initialiser son attribut spécifique. Le constructeur de la classe `Mecanisme` doit impérativement invoquer le constructeur de la super-classe `Produit`

notes

résumé

3m 37s



```
class MecanismeAnalogique extends Mecanisme {
    private int date;
    public MecanismeAnalogique(double valeurDeBase, String uneHeure, int uneDate) {
        super (valeurDeBase, uneHeure);
        date = uneDate;
    }
}
class MecanismeDigital extends Mecanisme {
    private String reveil;
    public MecanismeDigital(double valeurDeBase, String uneHeure, String heureRev) {
        super(valeurDeBase, uneHeure);
        reveil = heureRev;
    }
}
// ...
```

au travers de cette tournure. Si l'on veut, à la construction, attribuer une valeur par défaut à l'attribut heure alors on procédera, comme nous l'avons déjà fait précédemment pour la classe Produit, en surchargeant le constructeur, c'est-à-dire que nous définirons un second constructeur qui ne prendrait pas de paramètres pour l'heure du mécanisme mais qui initialiserait cet attribut au moyen d'une valeur spécifique.

notes

résumé

4m 25s




```
class MecanismeDouble extends MecanismeAnalogique {  
    private String reveil;  
    public MecanismeDouble(double valeurDeBase, String uneHeure,  
                           int uneDate, String heureReveil) {  
        super(valeurDeBase, uneHeure, uneDate);  
        reveil = heureReveil;  
    }  
}
```

Passons maintenant aux constructeurs des sous-classes. Tout d'abord, celui de la sous-classe `MecanismeAnalogique` par exemple, qui hérite directement de la classe `Mecanisme`. Ce constructeur prendra en paramètre des valeurs lui permettant d'initialiser l'ensemble de ses attributs, ceux hérités de plus haut et ceux qui lui sont spécifiques. Le constructeur de la sous-classe `MecanismeAnalogique` doit donc invoquer également le constructeur de la super-classe `Mecanisme` toujours par la même tournure. Et le constructeur de la classe `MecanismeDigital` se rédige de façon tout à fait analogue. Un constructeur possible pour la classe `MecanismeDouble` est un constructeur permettant

notes

résumé

4m 50s



Pour le moment, il n'y a qu'un constructeur par défaut, mais fixons la construction des `Mecanisme` :

- ▶ initialisation de la valeur de base (`Produit`)
- ▶ initialisation de l'heure

```
class Mecanisme extends Produit {  
    private String heure;  
  
    public Mecanisme(double valeurDeBase, String uneHeure) {  
        super(valeurDeBase);  
        heure = uneHeure;  
    }  
    public Mecanisme(double valeurDeBase){  
        super(valeurDeBase);  
        heure = "12:00";  
    }  
}
```

d'initialiser l'ensemble de ses attributs, ceux qui lui sont spécifiques comme ici le réveil, comme ceux qui sont hérités de plus haut. On va hériter de plus haut plusieurs attributs la date du `MecanismeAnalogique`, l'heure du `Mecanisme` et la valeur de base du `Produit`. Ce constructeur de `MecanismeDouble` appellera donc le constructeur de la super-classe pour initialiser les attributs hérités de plus haut et initialisera son attribut propre. Intéressons-nous maintenant à la gestion des valeurs par défaut par les constructeurs. Rappelez-vous que lorsque nous avons défini les constructeurs de la classe `Mecanisme`, nous avons donné la possibilité de construire un mécanisme sans fournir d'heure, auquel cas l'heure aurait une valeur par défaut. Et nous nous posons maintenant la question de savoir comment les constructeurs des sous-classes peuvent préserver cette valeur par défaut.

notes

résumé

5m 25s



```
class MecanismeDouble extends MecanismeAnalogique {  
  
    public MecanismeDouble(double valeurDeBase, String uneHeure, int uneDate,  
                           String heureReveil) {  
        super(valeurDeBase, uneHeure, uneDate);  
        reveil = heureReveil;  
    }  
  
    // gestion propre de la valeur par défaut de l'heure (super-classe)  
    public MecanismeDouble(double valeurDeBase, int uneDate,  
                           String heureReveil) {  
        super(valeurDeBase, uneDate);  
        reveil = heureReveil;  
    }  
}
```

On souhaiterait donc pouvoir par exemple

notes

résumé

6m 13s



construire un `MecanismeDouble` sans fournir d'heure et à ce moment-là on voudrait que son attribut `heure` ait la même valeur par défaut que celle qui est prévue pour les mécanismes tout court et l'on veut bien sûr garder la possibilité d'initialiser son attribut spécifique. La solution passe à nouveau par la surcharge. La surcharge définirait le constructeur de la classe `MecanismeDouble` de sorte à ce qu'il puisse fonctionner sans avoir de valeur pour son attribut `heure`. Il appellerait alors un constructeur de la super-classe programmé sur le même principe, c'est-à-dire un constructeur qui n'aurait pas besoin d'une heure comme paramètre. Ceci veut dire que dans la super-classe `MecanismeAnalogique`, il existe un constructeur qui se passe d'un paramètre pour l'heure et sauriez-vous dire quelle va être la première instruction de ce constructeur ? première instruction de ce constructeur ?

6m 14s

