

Support de cours

Cours:

Introduction à la programmation orientée objet (en C++)

Vidéo:

W17-01-pbgeneral-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Différents mécanismes. Mécanisme digital. Mécanisme analogique. Mécanismes doubles. Montres analogiques. Montres digitales. Modélisation du problème d'un point. Mécanismes de base. Niveau du produit. Notion de conception du problème. Affichage polymorphique. Classes abstraites. Notion de pointeurs. Logiciel de gestion de bijouterie. Vue programmation.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en C++).)



[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 : présentation et modélisation du problème

(Partie 1)

Introduction à la programmation orientée objet (en C++)

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Voici donc la dernière série de séquence vidéo de ce mooc

notes

résumé

0m 1s



Problématiques abordées :

- ▶ conception POO, héritage, classes abstraites (prix, affichage)
- ▶ affichage polymorphique
- ▶ surcharge d'opérateurs
- ▶ héritage multiple
- ▶ copie polymorphique

sur l'introduction à la programmation orientée objet en C++. Dans cette dernière série de séquence vidéo, nous avons voulu vous présenter un problème dans sa globalité, un problème très général, vous présenter une étude de cas, où il s'agira donc de présenter différentes montres, des montres analogiques, des montres à aiguille, des montres digitales, des montres qui seront les deux, les montres auront des accessoires comme des bracelets, des boîtiers, etc. Donc un cadre assez général, qui pourrait par exemple servir à un logiciel de gestion de bijouterie, ou un logiciel de vente, ou même d'impressions de catalogues, dans le cadre duquel nous allons pouvoir vous illustrer différents concepts présentés tout au long de ce cours.

notes

résumé

0m 5s



Problématiques abordées :

- ▶ conception POO, héritage, classes abstraites (prix, affichage)
- ▶ affichage polymorphique
- ▶ surcharge d'opérateurs
- ▶ héritage multiple
- ▶ copie polymorphique

Les thématiques que nous avons choisi de vous présenter dans ces vidéos de synthèse contiennent tout d'abord bien sûr la notion de conception du problème, la modélisation du problème d'un point de vue programmation orienté objet, quelles classes doivent hériter de quelles autres, est-ce qu'il doit y avoir des classes abstraites,

notes

résumé

0m 46s



Problématiques abordées :

- ▶ conception POO, héritage, classes abstraites (prix, affichage)
- ▶ affichage polymorphique
- ▶ surcharge d'opérateurs
- ▶ héritage multiple
- ▶ copie polymorphique

comment par exemple rendre le calcul du prix

notes

résumé

1m 1s



Problématiques abordées :

- ▶ conception POO, héritage, classes abstraites (prix, affichage)
- ▶ affichage polymorphique
- ▶ surcharge d'opérateurs $+=$
- ▶ ~~héritage~~ multiple
- ▶ copie polymorphique

des montres polymorphiques, ou l'affichage polymorphique, L'affichage justement sera traité de façon très spécifique avec la surcharge de l'opérateur d'affichage, et puis révision de la façon de faire un affichage qui s'adapte à chacune des classes, un affichage polymorphique. Nous avons aussi introduit la surcharge d'opérateurs au travers d'un opérateur $+=$, qui nous permettra de rajouter des composants aux montres que l'on voudra constituer. Nous aborderons également l'héritage multiple

notes

résumé

1m 2s



Le problème

- ▶ Les *montres* sont des *produits* (que l'on peut vendre : ont un prix)
- ▶ Les montres ont un *mécanisme* de base et sont constituées de différents *accessoires* (boîtier, bracelet, ...)
- ▶ Les *produits* ont un prix dont le calcul **peut varier**, à partir d'une valeur de base
- ▶ Tous les produits sont « **affichables** », chacun à sa façon
- ▶ Les *mécanismes* et *accessoires* de montre sont aussi des produits (on pourrait en acheter séparément)
- ▶ Il existe trois sortes de *mécanismes* : *analogiques* (montre à aiguilles), *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*

avec les différents mécanismes, mécanisme digital ou mécanisme analogique, à aiguille, voire les mécanismes doubles, qui sont à la fois à aiguille et digitales, et puis nous terminerons donc par une dernière séquence vidéo, qui présentera le problème non encore abordé dans le cours pour l'instant, de la notion de copie polymorphique, comment copier des collections hétérogènes, comment allons-nous copier les montres qui contiennent différents composants, qui peuvent se comporter de façon différente, soit du point de vue du prix, soit de l'affichage. Voilà donc le menu de cette dernière semaine. Mais commençons par présenter le problème un petit plus en détails, ce que nous voulons donc modéliser ce sont des montres, on va dire que des montres sont des produits, au sens qu'un produit est quelque chose que l'on peut vendre, qui a un prix. Les montres par ailleurs auront des mécanismes de base, typiquement donc des aiguilles pour pouvoir afficher l'heure, et seront constituées de différents accessoires, comme un boîtier, un bracelet, le verre sur la montre, le fermoir... Les produits dont nous avons parlé tout à l'heure ont un prix, et le calcul de ce prix peut varier, donc cet aspect là est intéressant pour la conception, à partir d'une valeur de base quoi doit être fixée au niveau du produit. Tous les produits sont affichables, et cet affichage peut aussi varier, chaque produit doit être affiché à sa façon propre. Bien sûr ce qui est derrière ces notions de « peut varier » et « affichable à sa façon propre », c'est la notion de polymorphisme. Ensuite les mécanismes dont nous avons parlé juste au dessus, et les accessoires dont sont constituées les montres,

notes

résumé

1m 32s





ces mécanismes et ces accessoires sont aussi des produits. On voit ici donc l'emploi plusieurs fois du verbe être, qui va nous qualifier donc des relations d'héritage. On pourrait par exemple acheter séparément des accessoires, des mécanismes, donc chacun pourrait se comporter comme un produit, donc chacun pourrait avoir son prix propre, et sa façon propre de calculer son prix. Il existe fondamentalement trois sortes de mécanismes, des mécanismes que l'on va qualifier d'analogiques, pour représenter donc les montres à aiguilles, les montres digitales seront représentées par un mécanisme digital, et puis on aura des montres qui seront à la fois à aiguille et à la fois à affichage digital. Enfin pour ces mécanismes que l'on a qualifiés de « mécanismes doubles », on supposera qu'ils n'indiquent qu'une seule heure. Donc on va dire qu'il n'y aura qu'une heure associée à une montre ici, et qu'elle aura deux façons de se représenter. C'est un point de vue choisi pour l'exercice,

notes

résumé

3m 13s



Le problème

- ▶ Les *montres* sont des *produits* (que l'on peut vendre : ont un prix)
- ▶ Les montres ont un *mécanisme* de base et sont constituées de différents *accessoires* (boîtier, bracelet, ...)
- ▶ Les *produits* ont un prix dont le calcul peut varier, à partir d'une valeur de base
- ▶ Tous les produits sont « affichables », chacun à sa façon
- ▶ Les *mécanismes* et *accessoires* de montre sont aussi des produits (on pourrait en acheter séparément)
- ▶ Il existe trois sortes de *mécanismes* : *analogiques* (montre à aiguilles), *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*

évidemment on aurait pu choisir un autre point de vue, qui consistait à associer une heure à chacun des mécanismes de représentation, soit aiguille, soit digital, mais ce n'est pas le point de vue qu'on a pris ici, justement pour pouvoir introduire une problématique intéressante au niveau du codage de ces classes.

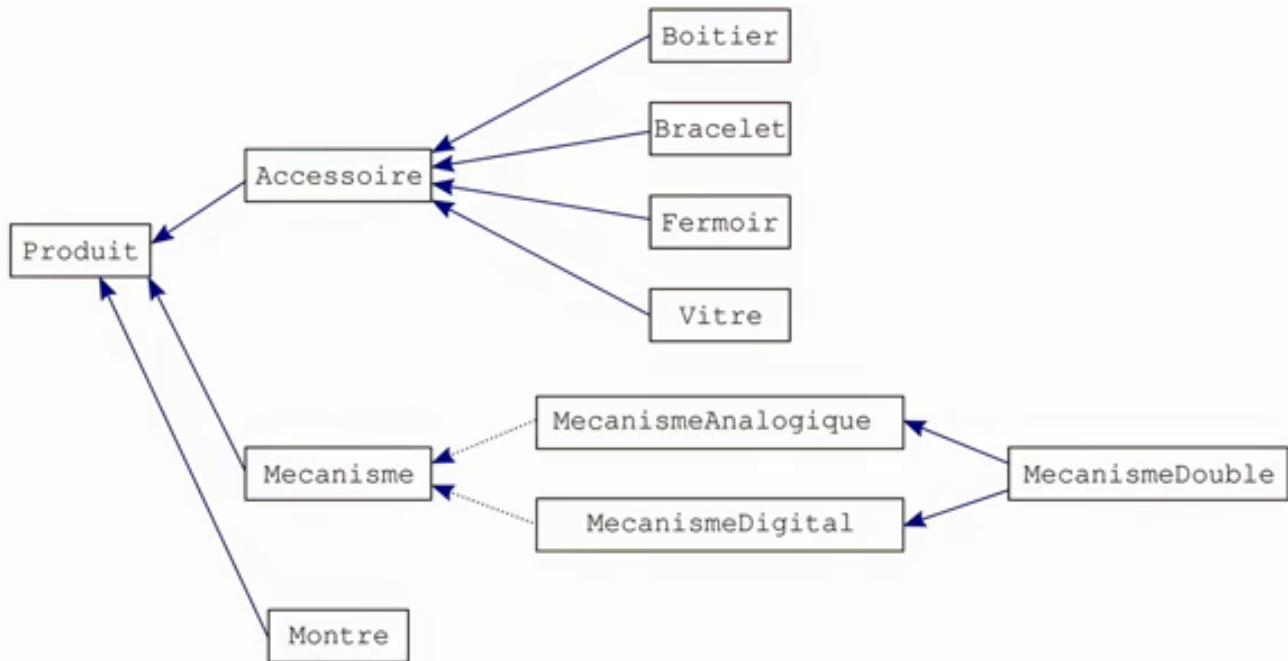
notes

résumé

4m 9s



Hierarchie de classes



Pour résumer, au niveau des classes, c'est-à-dire au niveau des types d'objets que nous aurons dans notre programme, nous allons avoir donc des montres, des produits, des mécanismes, et des accessoires, les accessoires peuvent être des boîtiers, des bracelets, tous ceci seront des classes, au niveau des mécanismes, on peut avoir des mécanismes analogiques, des mécanismes digitaux, des mécanismes doubles. Voilà l'ensemble des classes. Au niveau des relations donc d'héritage, nous avons vu que les montres sont des produits, donc vont hériter de produits. On a vu que les mécanismes analogiques, digitaux et doubles, sont des sortes de mécanismes, donc les trois classes ici vont hériter naturellement de la classe « Mecanisme », nous avons aussi vu que des mécanismes et les accessoires sont également des produits. Tout ceci, donc, va nous conduire à la modélisation hiérarchique suivante : nous avons donc tout en haut la notion de produit, les accessoires, les mécanismes, les montres, sont des produits. Par contre, une montre aura un mécanisme et aura des accessoires, donc encapsulera ces classes là. Parmi les accessoires, nous aurons des boîtiers, des bracelets, des fermoirs, des vitres, tous ceci sont des accessoires.

notes

résumé

4m 25s



Le problème

- ▶ Les *montres* *sont des produits* (que l'on peut vendre : ont un prix)
- ▶ Les montres ont un *mécanisme* de base et sont constituées de différents *accessoires* (boîtier, bracelet, ...)
- ▶ Les *produits* ont un prix dont le calcul *peut varier*, à partir d'une valeur de base
- ▶ Tous les produits sont « *affichables* », chacun à sa façon
- ▶ Les *mécanismes* et *accessoires* de montre *sont* aussi des produits (on pourrait en acheter séparément)
- ▶ Il existe trois sortes *de mécanismes* : *analogiques* (montre à aiguilles), *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*

Au niveau des mécanismes, les mécanismes analogiques et les mécanismes digitaux sont des mécanismes. Enfin, le « mécanisme double », au départ c'est un mécanisme, mais nous avons décidé par la dernière remarque précédente,

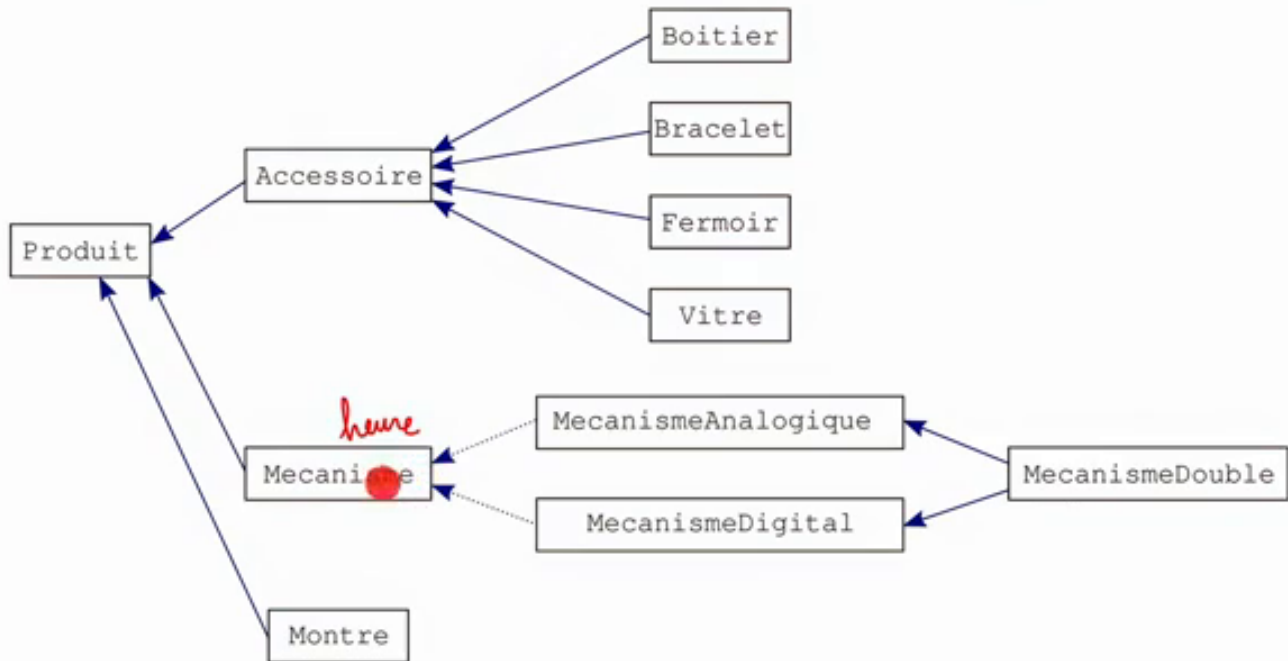
notes

résumé

5m 47s



Hierarchie de classes



qu'il n'indiquait qu'une seule heure, qu'on va certainement représenter au niveau de la notion de mécanisme, et qu'ils se comportent à la fois comme des mécanismes analogiques et des mécanismes digitaux, d'où naturellement on va dire, qu'un mécanisme double est un mécanisme analogique, et est en même temps un mécanisme digital, tout en étant une seule fois, c'était ça la remarque sur « ne comportera qu'une seule heure », tout en étant donc qu'une seule fois un mécanisme, et non pas deux fois, un mécanisme, c'était la contrainte au niveau de la conception qu'on a voulu vous imposer,

notes

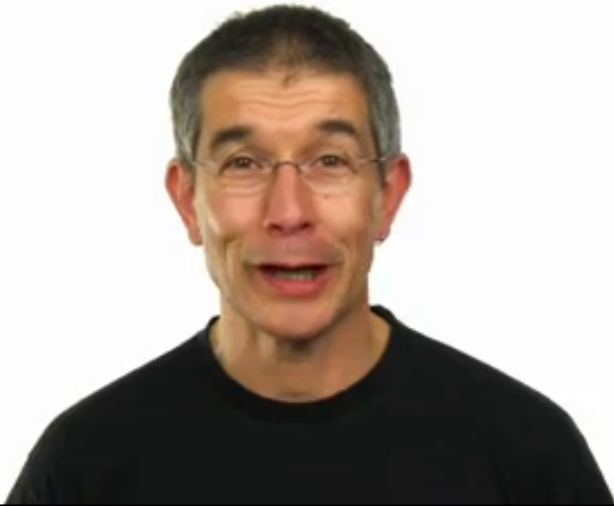
résumé

5m 59s



Du problème au premier code

- Les *montres* sont des *produits*



```
class Produit {  
};  
  
// =====  
class Montre : public Produit {  
};  
  
// =====  
int main() {  
    return 0;  
}
```

pour avoir justement ici un héritage virtuel, faire de « Mekanisme » une classe virtuelle.

notes

résumé

6m 37s



Du problème au premier code

- Les *montres* sont des *produits*

```
class Produit {  
};  
  
// -----  
class Montre : public Produit {  
};  
  
// -----  
int main() {  
  
    return 0;  
}
```

Regardons comment tout ceci se traduit au niveau du code. Nous vous conseillons d'ailleurs en même temps que vous suivez ces vidéos, de temps en temps faire des pauses, et écrire par vous-même le code dans votre environnement de développement habituel.

notes

résumé

6m 46s



Du problème au premier code

- Les *montres* sont des *produits*

```
class Produit {  
};  
  
// -----  
class Montre : public Produit {  
};  
  
// -----  
int main() {  
  
    return 0;  
}
```

Le premier point : les montres sont des produits, se traduit donc par une classe « Montre », qui est un, qui hérite d'un produit, et donc bien sûr, on introduit une classe « Produit », dont hérite « Montre », et puis ensuite, pour l'instant, on a simplement un « main » vide.

notes

résumé

6m 57s



Du problème au premier code

- ▶ Les *montres* sont des *produits*
- ▶ Les montres ont un *mécanisme* et sont constituées de différents *accessoires*

```
#include <vector>
using namespace std;

// ...

// =====
class Accessoire {
};

// =====
class Mecanisme {
};

// =====
class Montre : public Produit {
private:
    Mecanisme coeur;
    vector<Accessoire> accessoires;
};
```

Deuxième point, les montres ont un mécanisme, donc elles vont encapsuler un mécanisme, les montres ont un mécanisme, et sont constituées, donc ont, différents accessoires. Donc, comment on dit « différents accessoires », c'est simplement une collection d'accessoires, un tableau dynamique d'accessoires. Ceci introduit donc deux nouvelles classes,

notes

résumé

7m 13s



Du problème au premier code

- ▶ Les *montres* sont des *produits*
- ▶ Les montres ont un *mécanisme* et sont constituées de différents *accessoires*

```
#include <memory>
#include <vector>
using namespace std;
// ...

// =====
class Accessoire {
};

// =====
class Mecanisme {
};

// =====
class Montre : public Produit {
private:
    unique_ptr<Mecanisme> coeur;
    vector<unique_ptr<Accessoire>> accessoires;
};
```

une classe « Mecanisme » et une classe « Accessoire ». Ceci dit, nous savons que ces classes « Accessoire » et « Mecanisme » sont des classes très générales, nous les avons voulues très générales pour qu'elles puissent se comporter de façon polymorphique, on aura plusieurs accessoires, les bracelets... nous aurons plusieurs mécanismes, et donc ici nous voulons certainement du polymorphisme, et donc on va devoir ici rajouter la notion de pointeurs sur des mécanismes, et de pointeurs sur des accessoires, ce que nous faisons donc tout de suite en choisissant dans cette version ici, d'utiliser des pointeurs C++ 2011, des pointeurs dits intelligents, des « smart pointers », donc ici le « unique pointer » sur mécanismes, et un « unique pointeur » sur accessoires, ce qui nécessite donc que l'on inclue la bibliothèque memory pour les unique pointer. Bien sûr, on avait inclus ici la bibliothèque vector ici, pour les tableaux dynamiques dont nous avons besoin ici. Et comme nous venons d'introduire des pointeurs dans la classe « Montre », nous devons avoir le réflexe en tant que programmeurs, de penser à, de penser à,

notes

résumé

7m 37s

