

Support de cours

Cours:

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

Vidéo:

W17-04-heritmultipCPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

Construction des mécanismes. Méthode d'affichage générale. Constructeurs de sous-classe. Hiérarchie de classe. Affichage du cadran. Classe virtuelle. Valeur de base du produit. Type du mécanisme. Heure du mécanisme. Constructeur de cette classe virtuelle. Méthode d'affichage du type. Attribut spécifique. Méthode d'affichage du cadran. Classe virtuelle toutes. Constructeurs des sous-classes.



[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 : modélisation des mécanismes

(Partie 3)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Conséquences

En raison de la classe virtuelle `Mecanisme`, toutes les sous-classes **doivent** appeler le constructeur de cette classe !

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 : public Produit {
public:
    Mecanisme(double valeur_de_base, string une_heure = "12:00")
        : Produit(valeur_de_base), heure(une_heure)
    {}

private:
    string heure;
};
```

Dans une hiérarchie de classe ne comportant pas de classe virtuelle

notes

résumé

0m 1s



```
class MecanismeAnalogique : virtual public Mecanisme {
public:
    MecanismeAnalogique(double valeur_de_base, string une_heure, int une_date)
    : Mecanisme(valeur_de_base, une_heure), date(une_date)
    {}
private:
    int date;
};
// ...
class MecanismeDouble : public MecanismeAnalogique , public MecanismeDigital {
public:
    MecanismeDouble(double valeur_de_base, string une_heure, int une_date,
                     string heure_reveil)
    : Mecanisme(valeur_de_base, une_heure)
    , MecanismeAnalogique(valeur_de_base, une_heure, une_date)
    , MecanismeDigital(valeur_de_base, une_heure, heure_reveil)
    {}
};
```

tous constructeurs de sous-classe est simplement tenu d'invoquer les constructeurs de ces super-classes directes. Par contre dans une hiérarchie comportant une classe virtuelle toutes les sous-classes doivent appeler le constructeur de cette classe virtuelle. Commençons donc, par fixer la construction des mécanismes, jusqu'ici nous n'avions qu'un constructeur par défaut. Affinons un 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. On peut également imaginer de donner une valeur par défaut à ce second paramètre. Le constructeur de la classe « Mecanisme » doit naturellement invoquer le constructeur de la super-classe « Produit » et initialiser son attribut propre. 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 ces attributs,

notes

résumé

0m 5s



Conséquences

En raison de la classe virtuelle `Mecanisme`, toutes les sous-classes **doivent** appeler le constructeur de cette classe !

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 : public Produit {
public:
    Mecanisme(double valeur_de_base, string une_heure = "12:00")
        : Produit(valeur_de_base), heure(une_heure)
    {}

private:
    string heure;
};
```

ceux hérités de plus haut et ceux qui lui sont spécifiques et il va de toute façon invoquer le constructeur de sa super-classe directe qui se trouve être la classe virtuelle « `Mecanisme` ». L'écriture du constructeur dans la classe « `MecanismeDigital` » se fera de façon tout à fait analogue. Le constructeur de la sous-classe « `MecanismeDouble` » doit lui invoquer les constructeurs de ses super-classes directes mais doit d'abord invoquer le constructeur de la super-classe virtuelle. Vous souvenez vous de ce qu'il se passe avec l'appel au constructeur de la super-classe virtuelle dans les super-classes directes ?

notes

résumé

1m 25s



```
class MecanismeDouble : public MecanismeAnalogique , public MecanismeDigital {
public:
    MecanismeDouble(double valeur_de_base, string une_heure, int une_date,
                    string heure_reveil)
    : Mecanisme(valeur_de_base, une_heure)
    , MecanismeAnalogique(valeur_de_base, une_heure, une_date)
    , MecanismeDigital(valeur_de_base, une_heure, heure_reveil)
    {}

    MecanismeDouble(double valeur_de_base, int une_date, string heure_reveil)
    : Mecanisme(valeur_de_base)
    , MecanismeAnalogique(valeur_de_base, une_date)
    , MecanismeDigital(valeur_de_base, heure_reveil)
    {}
};
```

Intéressons nous maintenant à la gestion des valeurs par défaut. Rappelez vous que dans le constructeur de « Mecanisme » nous avons fait en sorte que le paramètre permettant d'initialiser l'heure du mécanisme est une valeur par défaut. Si nous souhaitons que cette heure par défaut soit préservée par les constructeurs des sous-classes de « Mecanisme » il faut prendre un certain nombre de mesure ensuite donc faire en sorte qu'il soit possible de construire un mécanisme analogique sans fournir d'heure et à ce moment là il aurait l'heure par défaut, la même que pour les mécanismes. Tout en ayant la possibilité bien sûr, d'initialiser son attribut spécifique. Nous devons pour cela prévoir un second constructeur qui ne prendrait pas de paramètre lié à l'heure et qui appellerait le constructeur de la super-classe, initialisant l'heure par défaut donc sans préciser de valeur pour cette heure. Mais n'aurait il pas été ici plus simple de mettre une valeur par défaut pour ce second paramètre ? La réponse est non car le dernier paramètre prévu pour la date n'a lui pas de valeur par défaut et toutes les valeurs par défaut des paramètres doivent nécessairement venir en fin de liste. Notez qu'il serait aussi très mauvais de dupliquer une même valeur par défaut à deux endroits différents par exemple ici, au niveau du mécanisme analogique et plus haut, au niveau du mécanisme. Ce serait la porte ouverte à des problèmes de mauvaise maintenance et d'incohérence.

notes

résumé

2m 1s



Supposons que l'on veuille :

- ▶ que tous les mécanismes s'affichent suivant le **même** schéma, **imposé** et non modifiable

Par exemple :

affichage du type de mécanisme, suivi d'un **affichage du cadran** (heure, date, heure de réveil, ...), suivi du prix

- ▶ mais que chaque **partie** de ce schéma soit adaptable
- ▶ offrir une version par défaut, utilisable dans les sous-classes, de l'affichage du cadran (par exemple affichage de l'heure)
- ▶ imposer la redéfinition de l'affichage du type de mécanisme

☛ Comment faire ?

notes

résumé

3m 26s



Supposons que l'on veuille :

- ▶ que tous les mécanismes s'affichent suivant le **même** schéma, **imposé** et non modifiable

Par exemple :

affichage du type de mécanisme, suivi d'un **affichage du cadran** (heure, date, heure de réveil, ...), suivi du prix

- ▶ mais que chaque **partie** de ce schéma soit adaptable
- ▶ offrir une version par défaut, utilisable dans les sous-classes, de l'affichage du cadran (par exemple affichage de l'heure)
- ▶ imposer la redéfinition de l'affichage du type de mécanisme

☛ Comment faire ?

auxquels elle s'applique. Le fait que le même schéma de base soit imposé à tous pour les mécanismes sous-entend que une fois cette méthode adhérant à ce schéma fixé, il ne faut plus qu'elle soit modifiée,

notes

résumé


```
class Mecanisme : public Produit {
public:
    //...
    // Tous les mécanismes DOIVENT s'afficher comme ceci
    virtual void afficher(ostream& sortie) const override final {
        sortie << "mécanisme "      ;    afficher_type(sortie);
        sortie << " (affichage : "  ;    afficher_cadran(sortie);
        sortie << " ), prix : "    ; Produit::afficher(sortie);
    }

protected: // On veut offrir la version par défaut aux sous-classes
    // Par défaut, on affiche juste l'heure.
    virtual void afficher_cadran(ostream& sortie) const {
        sortie << heure;
    }

private:
    virtual void afficher_type(ostream& sortie) const = 0;
};
```

notes

résumé

5m 37s



```
class Mecanisme : public Produit {
public:
    //...
    // Tous les mécanismes DOIVENT s'afficher comme ceci
    virtual void afficher(ostream& sortie) const override final {
        sortie << "mécanisme " ; afficher_type(sortie);
        sortie << " (affichage : " ; afficher_cadran(sortie);
        sortie << ")", prix : " ; Produit::afficher(sortie);
    }

protected: // On veut offrir la version par défaut aux sous-classes
    // Par défaut, on affiche juste l'heure.
    virtual void afficher_cadran(ostream& sortie) const {
        sortie << heure;
    }

private:
    virtual void afficher_type(ostream& sortie) const = 0;
};
```

» de la classe « Mecanisme » redéfinit la méthode « afficher » de la classe « Produit » nous avons prit le soin de la marquer comme « override ». La méthode « afficher_cadran », telle que stipulée dans les contraintes que l'on a énoncées tout à l'heure, offre une version par défaut qui permet d'afficher l'heure. Par contre la méthode « afficher_type », on veut en imposer la définition concrète dans les sous-classes mais elle reste quelque chose d'abstrait au niveau des mécanismes elle est donc déclarée comme une méthode virtuelle pure. Si l'on veut permettre aux sous-classes de mécanisme qui aurait redéfini « afficher_cadran » d'invoquer quand même la méthode « afficher_cadran » héritée de la super-classe alors il faut que l'accès à « afficher_cadran » soit possible dans ces sous-classes et c'est la raison pour laquelle nous avons étiqueté cette méthode comme étant protégée. À contrario, la méthode « afficher_type » ne sera pas ré-invoquée dans les sous-classes de « Mecanisme »

notes

résumé

```
class MecanismeAnalogique: virtual public Mecanisme {
    // ...
protected:
    virtual void afficher_type(ostream& sortie) const override {
        sortie << "analogique";
    }

    virtual void afficher_cadran(ostream& sortie) const override {
        // On affiche l'heure (façon de base)...
        Mecanisme::afficher_cadran(sortie);
        // ...et en plus la date.
        sortie << ", date " << date;
    }
    // ...
};
```

elle est virtuelle pure et son corps. Par conséquent, il convient tout à fait de la déclarer en privé . Tous les mécanismes avec lesquels

notes

résumé

8m 25s



```
class MecanismeDouble: public MecanismeAnalogique, public MecanismeDigital {
    // ...
protected:
    virtual void afficher_type(ostream& sortie) const override {
        sortie << "double";
    }

    virtual void afficher_cadran(ostream& sortie) const override {
        // Par exemple...
        sortie << "sur les aiguilles : ";
        MecanismeAnalogique::afficher_cadran(sortie);
        sortie << ", sur l'écran : ";
        MecanismeDigital::afficher_cadran(sortie);
    }
};
```

on veut pouvoir concrètement travailler, on veut pouvoir doter une montre d'un cœur qui a un mécanisme analogique, doivent par conséquent impérativement redéfinir concrètement la méthode « afficher_type » justement pour pouvoir être instanciée. On imagine ici que pour la classe « MecanismeAnalogique », « afficher_type » affiche simplement le libellé analogique. La méthode « afficher_cadran » a, certes une définition de base dans la super-classe, mais peut parfaitement être redéfinie dans les sous-classes, par exemple ici on peut imaginer de la redéfinir dans la sous-classe « MecanismeAnalogique » de sorte à ce qu'elle affiche l'heure en rappelant la méthode héritée de la super-classe. Donc on rappelle ici la petite notation par le biais de résolution de portée qui permet d'appeler la méthode « afficher_cadran » de la classe « Mecanisme » mais qui en plus, va afficher la date. De façon analogue, la classe « MecanismeDouble » va donc aussi offrir une définition concrète de « afficher_type ». Elle peut aussi redéfinir la méthode « afficher_cadran » et bénéficier la méthode « afficher_cadran » héritée et de « MecanismeAnalogique » et de « MecanismeDigital ». On afficherait par ce biais-là les informations propres au « MecanismeAnalogique » à savoir l'heure et la date et par ce biais-là les informations propres au « MecanismeDigital » à savoir l'heure encore suivi du réveil.

notes

résumé

8m 31s



Test : un exemple de main()

```
// test de l'affichage des mécanismes
MecanismeAnalogique v1(312.00, 20141212);
MecanismeDigital    v2( 32.00, "11:45", "7:00");
MecanismeDouble     v3(543.00, "8:20", 20140328, "6:30");
cout << v1 << endl << v2 << endl << v3 << endl;

// Test des montres
Montre m(new MecanismeDouble(468.00, "9:15", 20140401, "7:00"));
m += new Bracelet("cuir", 54.0);
m += new Fermeoir("acier", 12.5);
m += new Boitier("acier", 36.60);
m += new Vitre("quartz", 44.80);
cout << endl << "Montre m :" << endl;
cout << m << endl;
```

Le code complet à ce stade (263 lignes) peut être téléchargé sur le site du cours :
<https://d396qusza40orc.cloudfront.net/cpp-fr/complements/final02.cc>

Et maintenant, comme nous en avons désormais l'habitude

notes

résumé

9m 49s



```
// test de l'affichage des mécanismes
MecanismeAnalogique v1(312.00, 20141212);
MecanismeDigital    v2( 32.00, "11:45", "7:00");
MecanismeDouble     v3(543.00, "8:20", 20140328, "6:30");
cout << v1 << endl << v2 << endl << v3 << endl;

// Test des montres
Montre m(new MecanismeDouble(468.00, "15", 20140401, "7:00"));
m += new Bracelet("cuir", 54.0);
m += new Fermeoir("acier", 12.5);
m += new Boitier("acier", 38.60);
m += new Vitre("quartz", 44.80);
cout << endl << "Montre m : " << endl;
cout << m << endl;
```

Le code complet à ce stade (263 lignes) peut être téléchargé sur le site du cours :
<https://d396qusza40orc.cloudfront.net/cpp-fr/complements/final02.cc>

nous pouvons tester les nouveaux développements au moyen d'un petit programme principal. Nous pouvons construire un mécanisme analogique en utilisant la valeur par défaut pour l'heure. C'est à dire en ne précisant rien pour l'heure. Nous pouvons construire un mécanisme digital qui lui aurait une autre valeur que la valeur par défaut pour l'heure et des valeurs spécifiques pour ces paramètres et de façon analogue, un « MecanismeDouble » qui donnerait toutes les informations nécessaires à son initialisation comme une heure ici, une date et l'heure du réveil. La méthode « afficher » des mécanismes que nous venons de développer va en fait être invoquée par la surcharge de l'opérateur d'affichage que nous avons développé précédemment pour les produits. Cette méthode est polymorphique et va donc s'adapter à tous types de produits donc y compris à des mécanismes. Ces lignes vont donc nous permettre de tester nos dernier développements sur les constructeurs dans la hiérarchie de « Mecanisme » et cette ligne va nous permettre de tester notre méthode d'affichage. Nous pouvons maintenant doter notre montre d'un cœur concret, nous pouvons par exemple imaginer de définir dans la classe « Montre » un constructeur qui prendrait en argument la valeur du pointeur vers un mécanisme. Donc on peut construire ce mécanisme par le biais des constructeurs que nous avons détaillés dans cette séquence. L'affichage de la montre serait lui aussi complété de sorte à invoquer l'affichage de son cœur. Le code complet de cette partie est téléchargeable sur le site du cours. Et ceci conclut cette séquence sur la modélisation des mécanismes sur la modélisation des mécanismes

notes

résumé

9m 50s

