

Support de cours

Cours:

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

Vidéo:

W17-05-copieprofonde-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Problème de la copie de montres. Copie polymorphique. Constructeur de copie. Problème de copie de montres. Copie de surface de m1. Pointer de façon unique. Copie de m1. Étude de cas. Copie profonde. Différentes valeurs de m1. Copie de surface. Constructeur de montre. Schéma suivant. Différents composants. Tableau dynamique de pointeurs.



[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 : copie profonde

(Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





notes

résumé

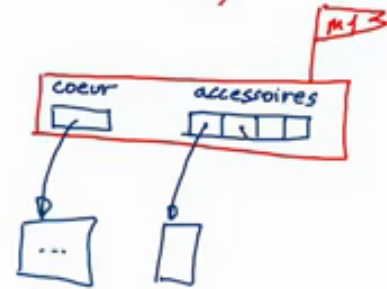
0m 1s



Copie profonde

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

Montre m1 (...);
Montre m2 (m1);



Si l'on veut faire des copies de **Montres**, on **doit** ici faire une **copie profonde** :
copie de chaque constituant (mécanisme, accessoires)

Il est alors également normal de redéfinir également l'opérateur d'affectation (=)

à effectivement comment copier une montre ce qui va nous amener à une nouvelle problématique qui est la copie polymorphique. Commençons par le problème de copie de montres et supposons que l'on ait, par exemple, deux montres m1 et m2 telles que m2 soit une copie de m1. Nous avons donc en mémoire le schéma suivant : le cœur qui est un pointeur sur un mécanisme quelque part en mémoire et les accessoires qui sont donc, un tableau dynamique de pointeurs

notes

résumé

0m 25s



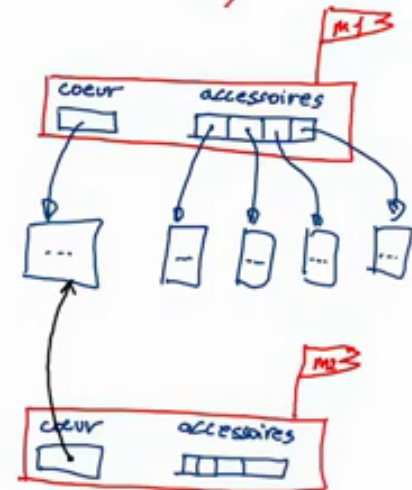
Copie profonde

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

Si l'on veut faire des copies de **Montres**, on **doit** ici faire une **copie profonde** : copie de chaque constituant (mécanisme, accessoires)

Il est alors également normal de redéfinir également l'opérateur d'affectation (=)

Montre m1 (...);
Montre m2 (m1);



vers différents accessoires en mémoire. Si l'on fait une copie de surface de m1 dans m2, copie de surface, c'est à dire que l'on va simplement recopier les différentes valeurs de m1, valeurs de pointeurs,

notes

résumé

1m 1s

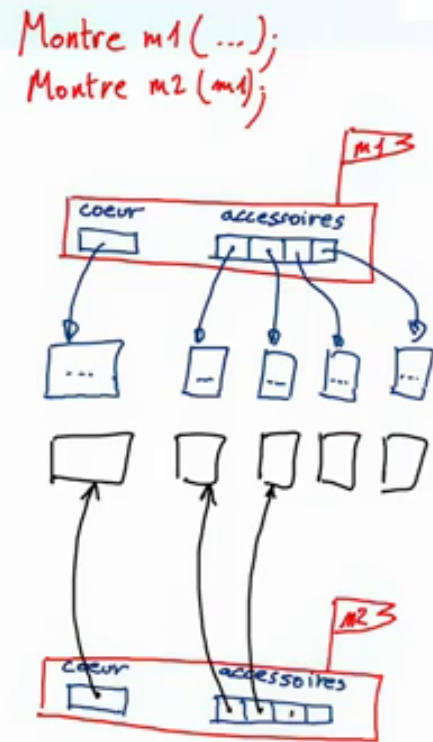


Copie profonde

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

Si l'on veut faire des copies de **Montres**, on **doit** ici faire une **copie profonde** : copie de chaque constituant (mécanisme, accessoires)

Il est alors également normal de redéfinir également l'opérateur d'affectation (=)



donc ici, on recopiera le pointeur vers cet élément et cetera, alors à ce moment là, si ceci était possible, je vais revenir dans un instant sur ce point là, on aurait, de toute façon déjà un problème puisque la montre m2 et la montre m1 partagent les mêmes accessoires et partagent le même mécanisme ce qui fait que si, au travers de m1 on remet, par exemple, sa montre à l'heure, automatiquement, la montre de votre ami, qui est la montre m2, elle aura le même heure. Si vous décidez de changer le bracelet de la montre m1 pour avoir un bracelet en cuir, automatiquement, la montre m2 aura aussi un bracelet en cuir. Qui plus est, avec l'architecture que l'on a défini actuellement, c'est à dire, avec des unique_ptr, on ne peut pas de toute façon faire ces copies puisque les unique_ptr doivent pointer de façon unique vers les zones mémoires, on ne peut pas avoir de copie de unique_ptr. De toute façon, on ne pourrait pas faire ce schéma en mémoire et si on avait ici décidé d'avoir des pointeurs à la 'c' alors on aurait le problème que j'ai évoqué précédemment, d'avoir des composant qui seraient partagés par différentes montres. Donc la conclusion de tout ceci c'est qu'il faut faire une copie profonde, copie profonde qui signifie qu'au lieu de simplement copier les valeurs de pointeurs, copier les adresses on copie les objets pointés eux mêmes et donc à chaque composant on effectue effectivement une copie de l'objet en question et on stock dans l'objet copié au plus haut niveau, les pointeurs

notes

résumé

1m 25s



Constructeur de copie

```
Montre(const Montre& autre)
// Ne pas oublier d'appeler le constructeur DE COPIE de la super-classe
: Produit(autre),
  coeur(???)
// ...??
{
  // ...??
}
```

vers les copies des composants. On a fait la copie, non pas au niveau superficiel en surface mais on a fait la copie en profondeur. Et on en a donc ici, non pas des copies des valeurs de pointeurs mais on a des pointeurs vers des copies des objets pointés. Voilà ce dont il s'agit quand on parle de copie profonde par opposition à une copie de surface et donc ici ce qu'il nous faut c'est bien une copie profonde, une copie de chaque constituant. Dans ces cas là il est d'ailleurs usuel d'également redéfinir l'opérateur d'affectation, ce que nous allons faire aussi dans un second temps. Commençons donc par le constructeur de copie c'est donc le constructeur de Montre qui fait une construction de copie donc le paramètre reçu par le constructeur de copie c'est bien sûr une autre montre, l'écriture de ce constructeur de copie doit commencer, il ne faut surtout pas oublier cet aspect là par appeler les constructeurs de copie des super-classes, ici on a une seule super-classe, la classe Produit. Et donc ici il faut appeler le constructeur de copie de la classe Produit, sinon... de la classe Produit, sinon...

notes

résumé

3m 2s

