

Support de cours

Cours:

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

Vidéo:

### W14-04-copieprofonde-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Constructeur de copie. Thème des constructeurs. Défaut du constructeur de copie. Copie de surface. Copie profonde. Constructeurs de copie. Fameuse classe. Façon explicite. Thème de cette séquence. Défaut des méthodes constructeur. Membre des attributs. Fait d'une copie membre. Attribut hauteur du rectangle. Méthodes particulières. Valeurs des attributs.**



[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/>



# Copie profonde

## (Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Lorsque nous avons abordé le thème des constructeurs, nous avons vu que ces méthodes étaient tellement importantes que C++ en fournissait une version par défaut. C'est le cas notamment du constructeur de copie. Nous avons vu également que la version par défaut du constructeur de copie faisait ce que l'on appelle une copie de surface, que cette façon de copier les objets est satisfaisante dans la plupart des cas

notes

résumé

0m 1s



## Petit rappel (2)

Dans certains cas, les versions minimales par défaut des méthodes constructeurs/destructeurs **ne sont pas adaptées** : exemple du **comptage des instances** (cf. semaine passée).

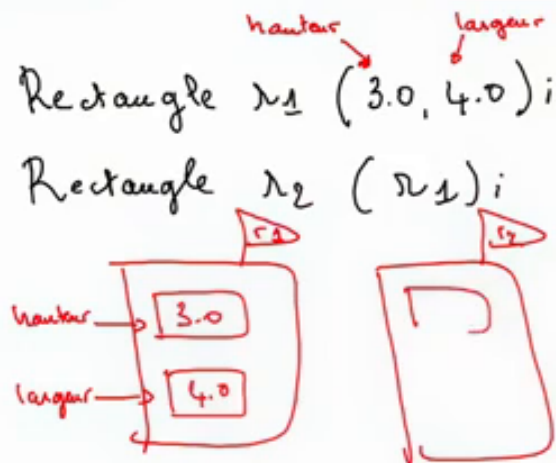
Autre exemple :

Le **constructeur de copie par défaut** réalise une copie membre à membre des attributs

→ **copie de surface**

Ceci pose typiquement problème lorsque **certain attributs** de la classe sont des **pointeurs**.

Examinons pourquoi sur un exemple concret...



mais pas tous, dans d'autres cas il faut avoir recours à ce que l'on appelle la copie profonde et c'est le thème de cette séquence. Pour rappel, nous savons donc qu'il existe en C++ des méthodes particulières permettant d'initialiser un objet en début de vie, « les constructeurs », de copier un objet dans un autre, « les constructeurs de copie », et de libérer les ressources associées à un objet, « les destructeurs ». C'est à une version minimale par défaut de ces méthodes que nous allons nous intéresser et en particulier dans le cadre de la copie. Nous avons vu dans une séquence précédente sur l'exemple du « comptage des instances » que les versions minimales par défaut des méthodes constructeur destructeur n'étaient pas toujours adaptées. Nous avons également vu que lorsqu'il était nécessaire d'en définir un de façon explicite, et bien, il fallait considérer la définition de tous les autres, également de façon explicite. Nous allons maintenant étudier un autre exemple lié au constructeur de copie par défaut qui réalise une copie de surface et nous allons voir que cette copie de surface peut poser un certain nombre de problèmes typiquement lorsque certains attributs sont des « pointeurs ».

Commençons par préciser un peu ce qu'est la copie de surface : il s'agit en fait d'une copie membre à membre des attributs. Supposons que l'on dispose de notre fameuse classe « Rectangle » et que dans un programme on déclare deux objets de type « Rectangle » un « Rectangle r1 » dont la largeur et la hauteur seraient initialisées au moyen d'un des constructeurs de la classe « Rectangle », et un second objet toujours de type « Rectangle », « r2 », obtenu par copie, initialisé par copie, à partir du rectangle « r1 ». Supposons que dans le constructeur utilisé le premier

### notes

### résumé

0m 25s



## Petit rappel (2)

Dans certains cas, les versions minimales par défaut des méthodes constructeurs/destructeurs **ne sont pas adaptées** : exemple du *comptage des instances* (cf. semaine passée).

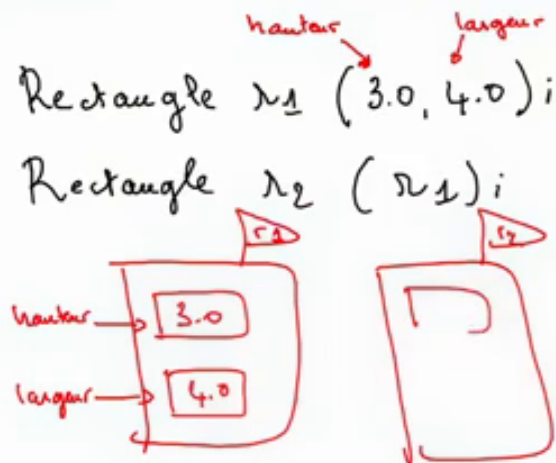
Autre exemple :

Le *constructeur de copie par défaut* réalise une copie membre à membre des attributs

→ **copie de surface**

Ceci pose typiquement problème lorsque **certain attributs** de la classe sont des **pointeurs**.

Examinons pourquoi sur un exemple concret...



argument correspond à la hauteur et le second à la largeur ; le rectangle « r1 » serait donc un objet en mémoire qui aurait deux attributs, « largeur » et « hauteur », initialisés comme suit. Supposons que dans cette classe « Rectangle » il n'existe aucune définition explicite du constructeur de copie, ceci signifie que au moment où l'on exécute cette ligne c'est le constructeur de copie par défaut qui est exécuté. Si l'on suppose donc que ceci correspond à l'attribut hauteur du rectangle « r1 » et ceci à son attribut « largeur », l'exécution du constructeur de copie par défaut va résulter dans la création d'un rectangle « r2 » dont les valeurs des attributs auraient été copiées membre à membre depuis ceux de « r1 » c'est-à-dire concrètement la valeur de l'attribut hauteur de « r2 » va être copiée depuis la valeur de l'attribut du même nom de « r1 » et pareil pour l'attribut « largeur ». Le constructeur de copie par défaut réalise donc une copie de surface, c'est-à-dire une copie membre à membre des attributs.

notes

résumé