

Support de cours

Cours:

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

Vidéo:

### W12-03-constrcopie-CPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

**Copie de surface. Constructeur de copie. Propre constructeur de copie. Plupart des cas. Règle d'or. Simulateur d'un monde physique. Opérateur d'affectation. Nom de méthode. Copie profonde. Opérateur égal. Référence constante. Prototype du constructeur de copie. Concepts destructeur. Nom de la classe. Copie de ce gros objet.**



[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/>

# Constructeur de copie

## (Partie 3)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Très souvent cette copie de surface suffit et c'est pour ça que vous n'aurez dans la plupart des cas pas besoin d'écrire explicitement un constructeur

notes

---

---

---

---

---

---

---

---

---

---

résumé

0m 1s



---

---

---

---

---

---

---

---

---

---

## Constructeur de copie (3)

- ▶ Un constructeur de copie est *automatiquement généré* par le compilateur s'il n'est pas explicitement défini (constructeur de copie par défaut)
- ▶ Ce constructeur opère une initialisation *membre à membre* des attributs (si l'attribut est un objet le constructeur de cet objet est invoqué)  
☞ **copie de surface**
- ▶ Cette copie de surface suffit dans la plupart des cas.

Cependant, il est parfois nécessaire de redéfinir le constructeur de copie, en particulier lorsque certains attributs sont des *pointeurs* (des exemples arriveront plus tard dans le cours) !

Mais je vous donne déjà la  **règle d'or**  : si vous touchez à l'un des trois parmi « constructeur de copie », « destructeur » et « opérateur d'affectation » (`operator=`), alors pensez aux deux autres.

de copie, mais il y a quand même des cas, en anticipant sur des exemples qui viendront bien plus tard dans le cours, où il sera nécessaire d'exprimer notre propre constructeur de copie. Il faudra faire ce que l'on appelle une copie profonde. Je vous donne déjà en anticipant aussi sur des concepts qui viendront par la suite, une règle d'or très importante, si vous avez à toucher à l'un des trois que sont le constructeur de copie, le destructeur et l'opérateur d'affectation, ce qu'on appelle l'opérateur égal, alors vous devez

notes

résumé

0m 13s





au moins vous posez la question de savoir s'il ne faut pas aussi redéfinir les deux autres. Voilà pour cette règle d'or qui anticipe sur des concepts destructeur,

notes

résumé

0m 37s



C++11 Par ailleurs, si l'on souhaite *interdire* la copie, il suffit de **supprimer** le constructeur de copie par défaut avec la commande « `= delete` » vue dans la séquence précédente.

Exemple :

```
class PasCopiable {  
    /* ... */  
    PasCopiable(PasCopiable const&) = delete;  
};
```



opérateur d'affectation, qui seront présentés plus tard dans le cours. Nous avons vu qu'un constructeur de copie par défaut est automatiquement fourni. Il peut aussi y avoir des cas où l'on souhaiterait ne pas autoriser la copie. Depuis C++ 2011 on peut donc explicitement demander à ne pas avoir de constructeur de copie. Cela peut par exemple être utile quand vous avez une classe qui contient beaucoup d'objets. Par exemple vous avez fait un simulateur d'un monde physique et vous ne souhaitez pas copier tout le monde avec tous ces objets donc vous voulez empêcher que l'on fasse une copie de ce gros objet qui contient beaucoup, beaucoup de choses. Cela se fait donc avec une syntaxe un peu particulière qui s'écrit donc `= delete`. Pour ça, il suffit donc de mettre `= delete` derrière le prototype du constructeur de copie. Par exemple, si on avait ici une classe que l'on ne souhaite pas copier, que je vais appeler ici `PasCopiable`, on aurait donc toutes les définitions comme d'habitude dans cette classe. Mais pour éviter la copie, on ajouterait un constructeur de copie, donc c'est bien un constructeur, ça a le même nom de méthode que le nom de la classe, ici ça reçoit un paramètre par référence constante qui est de même type, ici qui est de la même classe, donc `PasCopiable` aussi, mais simplement ce que l'on rajoute donc c'est que l'on rajoute ici à la fin du prototype `= delete`. Ça a pour effet qu'on ne peut pas à ce moment-là appeler

notes

résumé

0m 49s





le constructeur de copie puisqu'il a été détruit, delete, il n'existe plus. Donc cette case n'est pas copiable, on peut simplement l'initialiser avec les autres constructeurs éventuellement disponibles mais, en tout cas, ne pas créer de copie d'instance. Voilà, ceci conclut donc cette séquence sur les constructeurs de copie en C++. sur les constructeurs de copie en C++.

notes

résumé

2m 13s

