

Support de cours

Cours:

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

Vidéo:

W12-04-destructeurs-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Libération explicite des ressources. Affaiblissement de l'encapsulation. Méthodes publiques. Extérieur de la classe. Liste des paramètres. Méthode destructeur. Bonne solution. Méthodes particulières. Étude de cas. Fin de vie de l'instance. Source d'erreur. Opérations de libération nécessaires. Dehors de la classe. Corps du destructeur. Portions de mémoire.



[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/>

Destructeurs

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Destructeur

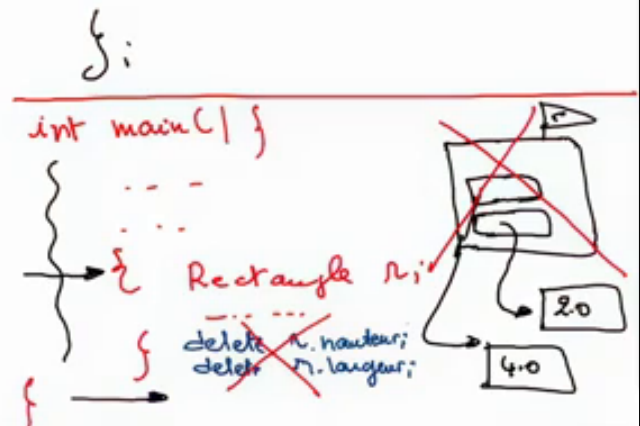
SI l'initialisation des attributs d'une instance implique la **mobilisation de ressources** : fichiers, périphériques, portions de mémoire (pointeurs), etc.

il est alors important de **libérer ces ressources** après usage !

Comme pour l'initialisation, l'*invocation explicite* de méthodes de libération n'est pas satisfaisante (fastidieuse, source d'erreur, affaiblissement de l'encapsulation).

C++ offre une méthode appelée **destructeur** invoquée automatiquement en fin de vie de l'instance.

```
class Rectangle {
public:
    Rectangle() : hauteur(new double(20)),
                 largeur(new double(40))
    {}
private:
    double* hauteur;
    double* largeur;
};
```



Alors ce n'est pas bon effectivement, essentiellement parce que c'est un affaiblissement de l'encapsulation ; si je veux avoir le droit de faire ceci, je ne peux plus mettre mes attributs en « private », Il faut qu'ils soient accessibles depuis l'extérieur de la classe, ou alors, l'alternative, je devrais programmer des méthodes publiques, comme un « getter » par exemple, qui me livrent le pointeur à libérer. Mais nous avons vu dans notre étude de cas sur le morpion que livrer un pointeur par le biais d'un « getter » était un affaiblissement de l'encapsulation quasiment aussi fort que de livrer directement l'attribut en le mettant en « public ». De plus, cette libération explicite des ressources depuis l'extérieur de la classe peut être source d'erreur (on peut oublier de libérer certaines ressources), et elle est fastidieuse : il faudrait le faire systématiquement. Le faire de cette façon n'est donc clairement pas la bonne solution. Pour que l'on se soit pas obligés d'explicitement libérer les ressources nécessitées par un objet en dehors de la classe, comme ceci, C++ offre des méthodes particulières que l'on appelle les destructeurs, qui ont la particularité d'être invoqués automatiquement en fin de vie de l'instance. Ce qui voudrait dire qu'ici, lorsqu'on termine ce bloc, la méthode destructeur serait automatiquement invoquée, permettant justement de libérer ces ressources sans casser l'encapsulation ni passer par les désavantages que nous avons vus ici. Il faut savoir que les ressources que l'on a besoin de libérer

notes

résumé

0m 1s



La syntaxe de déclaration d'un destructeur pour une classe `NomClasse` est :

```
~NomClasse() { // opérations (de libération) }
```

- ▶ Le destructeur d'une classe est une méthode *sans paramètre*
👉 **pas de surcharge possible**
- ▶ Son nom est celui de la classe, précédé du signe `~` (tilda).
- ▶ Si le destructeur n'est pas défini explicitement par le programmeur, le compilateur en génère automatiquement une version minimale.

en fin de vie d'un objet ne sont pas forcément des portions de mémoire, comme nous avons eu le cas dans le cadre de cet exemple, il peut s'agir d'autre chose, comme des fichiers ou des périphériques. Comment s'écrit concrètement un destructeur en C++ ? C'est une méthode dont le nom est le même que celui de la classe, mais précédé du symbole tilda (~), ce que nous voyons ici. La liste des paramètres est forcément vide et dans le corps du destructeur, nous allons mettre toutes les opérations de libération nécessaires en fin de vie d'un objet. Il faut noter qu'il n'y a qu'un seul destructeur possible dans la classe, donc pas de surcharge possible. Et si le destructeur n'est pas défini explicitement, alors le compilateur va en générer automatiquement une version minimale par défaut.

notes

résumé

1m 25s

