

Support de cours

Cours:

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

Vidéo:

W12-04-destructeurs-CPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

Méthode destructeur. Stade de l'exécution du programme. Petit exemple d'introduction. Attribut de type. Écriture de ma classe. Liste de paramètres vides. Version du destructeur. Extérieur du bloc. Destructeur. Corps vide. Opérations nécessaires. Cas. Classe. Fin de vie. Allocation dynamique des zones mémoires.



[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 3)

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 !

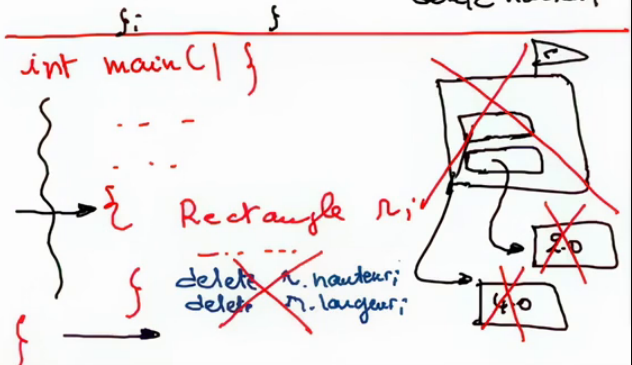
Comme pour l'initialisation, l'*invocation explicite* de méthodes de libération n'est pas satisfaisante (fastidieuse, source d'erreur, affaiblissement de l'encapsulation).

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))
    { } delete ?
```

```
private:
    double* hauteur;
    double* largeur;
```

```
public:
    ~Rectangle() { delete largeur;
                  delete hauteur; }
```



Pour revenir à notre petit exemple d'introduction, concrètement, comment faire pour programmer le destructeur ? Faisons un peu de place pour écrire notre destructeur, ici, je déclare une méthode destructeur, « tilda », même nom que celui de la classe, une liste de paramètres vides, et dans le corps, je me charge des opérations nécessaires pour libérer les ressources. Comme ceci. Et je termine ainsi l'écriture de ma classe. Ce destructeur est donc automatiquement invoqué lorsqu'une instance arrive en fin de vie. Typiquement, ici, lorsque nous arrivons à ce stade de l'exécution du programme, vu que cette variable « r » n'est plus définie à l'extérieur du bloc, le destructeur est invoqué, qui va être en charge de libérer les ressources liées à l'objet. Et ceci, sans avoir eu à casser l'encapsulation comme c'était le cas dans la solution vue précédemment. Si la classe « Rectangle » n'avait pas eu d'attribut de type « pointeur », et surtout si elle n'avait pas fait elle-même d'allocation dynamique des zones mémoires liées à ses attributs, il n'aurait pas été nécessaire de programmer explicitement un destructeur comme ici. Dans ce cas, le compilateur génère une version du destructeur par défaut, qui aurait un corps vide, donc nous n'aurions aucune instruction à l'intérieur. aucune instruction à l'intérieur.

notes

résumé

0m 1s

