

Support de cours

Cours:

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

Vidéo:

W14-03-1-heritageconstr-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Ordre d'appel des constructeurs. Constructeur d'une sous-classe. Attributs supplémentaires. Largeur égale. Premier temps. Corps du constructeur de la sous-classe. Hiérarchie d'héritages. Première partie. Certaine sorte de rectangle. Nouveaux attributs. Sous-classe de la classe. Seule taille. Ligne supplémentaire. Remarque finale. Seconde partie.



[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/>

Héritage : constructeurs (1)

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Encore un exemple

Il n'est pas nécessaire d'avoir des attributs supplémentaires...

```
class Carre : public Rectangle {
public:
    Carre(double taille)
        : Rectangle(taille, taille)
    {}
    /* Et c'est tout !
    (sauf s'il y avait des manipulateurs,
    il faudrait alors sûrement aussi les
    redéfinir)
    */
};
```

Rectangle
setHauteur(-) *largeur*
setLargeur(-) *hauteur*

Carre
void setHauteur(double l)
{
 hauteur = l;
}

Nous allons revenir en détails dans un instant sur l'ordre d'appel des constructeurs dans une hiérarchie d'héritages. Mais au préalable, je voudrais insister sur une remarque, il n'est pas nécessaire d'avoir des attributs supplémentaires dans la sous-classe ; par exemple, si nous avons un carré, qui est un rectangle, qui est une certaine sorte de rectangle, un carré est un rectangle qui a simplement la largeur égale à la hauteur, alors, on ne va pas introduire dans la sous-classe « Carre », d'attributs supplémentaires, mais on peut quand même, dans le constructeur de « Carre », et il faudrait le faire ici, on peut quand même, dans le constructeur d'une sous-classe qui n'introduit pas de nouveaux attributs, faire un appel, avoir besoin, de faire un appel, aux constructeurs de la super-classe. Regardons cet exemple en détails, on dit donc qu'un carré est un rectangle, dans la classe « Rectangle », nous avons comme d'habitude, les attributs largeur et hauteur, lesquels sont donc hérités dans la classe « Carre » et on dira simplement que carré est un rectangle où la largeur est égale à la hauteur. Et on ne rajoute rien de plus, on ne change rien d'autre à « Carre ». Donc, on va simplement déclarer la classe « Carre » comme étant une sous-classe de la classe « Rectangle », on ne va rien lui ajouter de particulier, je reviendrai avec une remarque finale dans un instant, mais simplement, on va forcer son constructeur à prendre ici une seule taille, la taille du côté d'un carré, et à faire appel ici au constructeur de la super-classe en demandant au constructeur de la super-classe d'avoir la hauteur et la largeur qui sont de même taille. Et on n'ajoutera rien dans le corps du constructeur de la sous-classe de la classe « Carre ». Voilà, et c'est tout simplement ces trois lignes, ici, on n'a rien besoin d'ajouter, elles permettront

notes

résumé

0m 1s



Encore un exemple

Il n'est pas nécessaire d'avoir des attributs supplémentaires...

```
class Carre : public Rectangle {
public:
    Carre(double taille)
        : Rectangle(taille, taille)
    {}
    /* Et c'est tout !
       (sauf s'il y avait des manipulateurs,
        il faudrait alors sûrement aussi les
        redéfinir)
    */
};
```

Rectangle
setHauteur(-) *largeur*
setLargeur(-) *hauteur*

Carre
void setHauteur(double l)
{
 hauteur = l;
}

d'avoir un carré qui est un rectangle dans lequel la construction oblige la largeur et la hauteur à être égales. Alors, quand on dit qu'il n'y a rien à ajouter, bien sûr, si au niveau de « Rectangle », nous avons une méthode qui serait par exemple une méthode « SetHauteur » ou une méthode « SetLargeur », alors ces deux méthodes seraient héritées par « Carre » et là, il faudrait bien sûr aussi les redéfinir pour obliger « SetHauteur » et « SetLargeur » à continuer de garder hauteur et largeur égales. Donc on pourrait par exemple faire quelque chose comme ceci, au niveau du « Carre », en spécialisant la méthode « SetHauteur », laquelle, dans « Carre », va masquer celle de « Rectangle », et qui, bien sûr, mettrait la hauteur... aux paramètres qu'elle a reçus, mais en plus, c'est ça qui est nouveau par rapport à « Rectangle ».

notes

résumé

1. Chaque constructeur d'une sous-classe *doit* appeler un des constructeurs de la super-classe
2. L'appel est la **1^{re} instruction**



en plus, obligerait la largeur à être égale à cette nouvelle hauteur. C'est donc cette ligne supplémentaire qui serait rajoutée dans la spécialisation de « Carre », et on ferait la même chose pour cette largeur, bien entendu.

notes

résumé

2m 49s



Et si l'on oublie l'appel à un constructeur de la super-classe ?

- ▶ Appel automatique au constructeur par défaut de la super-classe
- ▶ Pratique parfois, mais **erreur** si le **constructeur par défaut** n'existe pas

Rappel : le constructeur par défaut est particulier

- ▶ Il existe par défaut pour chaque classe qui n'a aucun autre constructeur
- ▶ Il disparaît dès qu'il y a un autre constructeur

Pour éviter des problèmes avec les hiérarchies **de** classes, dans un premier temps :

- ▶ Toujours déclarer au moins un constructeur
- ▶ Toujours faire l'appel à un constructeur de la super-classe

Pour résumer, à ce stade, chaque constructeur d'une sous-classe doit appeler un des constructeurs de la super-classe et cet appel doit se trouver en premier dans la liste d'initialisation. Que se passe-t-il si l'on oublie de faire explicitement appel à un des constructeurs de la super-classe ? Alors, le constructeur de la sous-classe appelle automatiquement le constructeur par défaut de la super-classe et évidemment, si la super-classe n'a pas de constructeur par défaut, il y aura une erreur de la part du compilateur. J'en profite pour vous rappeler que le constructeur par défaut, c'est-à-dire le constructeur qui ne prend pas d'arguments, est un peu particulier car si on ne l'écrit pas, on a par défaut une version fournie par le compilateur de ce constructeur par défaut, et que dès que l'on écrit un autre constructeur, alors le constructeur par défaut, par défaut, n'existe plus, et si on veut un constructeur par défaut, il faut à ce moment-là le réécrire. Donc, pour éviter tout problème lorsqu'on fait de l'héritage,

notes

résumé

3m 7s





dans un premier temps, je vous conseille de toujours déclarer au moins un constructeur et de toujours faire un appel explicite à un des constructeurs de la super-classe dans la sous-classe, même si c'est le constructeur par défaut que vous voulez appeler, vous pouvez quand même l'écrire explicitement, ce sera plus clair dans un premier temps. Voilà, ceci conclut cette première partie sur les conséquences de l'héritage sur les constructeurs, dans la seconde partie, nous reviendrons en détails sur l'ordre d'appel des constructeurs dans une hiérarchie de classes, puis nous vous donnerons encore quelques compléments. quelques compléments.

notes

résumé

4m 1s

