

Support de cours

Cours:

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

Vidéo:

W16-01-heritmultintro-CPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

Super-classes. Constructeur de la sous-classe. Ordre de déclaration des liens. Utilisation de l'héritage multiple. Moment de la déclaration de la classe. Troisième paramètre. Ordre d'appel des constructeurs. Fait analogue. Initialisation des attributs. Différents constructeurs. Constructeurs. Premier temps. Exemple concret. Liste d'initialisation. Plupart des compilateurs modernes.



[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 multiple (1) : concept et constructeurs

(Partie 3)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Syntaxe :

```
class nomSousClasse: public nomSuperClasse1, ...  
                    public nomSuperClasseN {  
  
//...  
};
```

Exemple :

```
class Ovovivipare: public Vivipare, public Vivipare {  
public:  
    Ovovivipare(unsigned int, unsigned int);  
    virtual ~Ovovivipare();  
protected:  
    bool espece_rare;  
};
```

L'**ordre** de déclaration des super-classes est pris en compte lors de l'invocation des constructeurs/destructeurs

Passons maintenant concrètement à l'utilisation de l'héritage multiple en C++ : si l'on veut déclarer en C++ qu'une sous-classe a plusieurs super-classes, il suffit au moment de la déclaration de la classe, d'indiquer l'ensemble des super-classes dont elle hérite, séparé par des virgules. Donc ici, pour chacune des super-classes dont hérite la sous-classe, il faudra indiquer « public "nom de la super-classe" » et on sépare les différentes super-classes par des virgules. Concrètement donc, ceci se lit « la classe « Ovovivipare » hérite

notes

résumé

0m 1s





de la super-classe « Ovipare » et de la super-classe « Vivipare » ». La classe contient ensuite un contenu tout à fait analogue à toutes les classes que vous avez écrites ou lues jusqu'ici. Notez pour finir un point à ne pas négliger : l'ordre de déclaration des liens d'héritage a son importance, il joue en effet un rôle dans la façon de construire ou détruire une instance bénéficiant de l'héritage multiple. Et c'est là le prochain point qui va nous occuper : comment fait-on de la construction ou destruction d'objets dans le cadre de l'héritage multiple Voyons donc ce que l'héritage multiple implique sur les constructeurs. Il n'y a en fait pas grand chose de nouveau, simplement un point sur

notes

résumé

0m 37s



Comme pour l'héritage simple, l'initialisation des attributs hérités doit être faite par invocation des constructeurs des super-classes :

Syntaxe :

```
SousClasse(liste de paramètres)
: SuperClasse1(arguments1),
...
  SuperClasseN(argumentsN),
  attribut1(valeur1),
...
  attributK(valeurK)
{ }
```



lequel il faut faire un peu attention.

notes

résumé

1m 13s



Comme pour l'héritage simple, l'initialisation des attributs hérités doit être faite par invocation des constructeurs des super-classes :

Syntaxe :

```
SousClasse(liste de paramètres)
: SuperClasse1(arguments1),
...
  SuperClasseN(argumentsN),
  attribut1(valeur1),
...
  attributK(valeurK)
{ }
```

Lorsque l'une des super-classes admet un constructeur par défaut, il n'est pas nécessaire de l'invoquer explicitement.

Donc, comme pour l'héritage simple, l'initialisation des attributs hérités des super-classes doit se faire dans la « liste d'initialisation » du constructeur de la sous-classe en appelant tous les constructeurs des super-classes dont on hérite. La syntaxe générale est donc la même que pour l'héritage simple. Dans la section « liste d'initialisation » du constructeur de la sous-classe, on appelle comme cela les différents constructeurs des super-classes dont on hérite, séparés par des virgules et, évidemment, si une des super-classes admet un constructeur par défaut,

notes

résumé

1m 15s





il n'est pas nécessaire de l'invoquer explicitement et, comme toujours,

notes

résumé

1m 49s





Attention ! L'exécution des constructeurs des super-classes se fait **selon l'ordre de la déclaration d'héritage**, et non selon l'ordre des appels dans le constructeur !

L'ordre des appels des destructeurs de super-classes est **l'inverse** de celui des appels de constructeurs



dans un premier temps, dans une première approche, où l'on apprend les constructeurs, je vous recommande de le faire pour vous souvenir qu'il y a bien, en effet, un appel au constructeur par défaut, même si ce n'est pas nécessaire.

notes

résumé

1m 53s



```
class Ovovivipare : public Ovipare, public Vivipare {
public:
    Ovovivipare(unsigned int nb_oeufs      ,
                unsigned int duree_gestation ,
                bool rarete                 = false );
    virtual ~Ovovivipare();
protected:
    bool espece_rare;
};

Ovovivipare::Ovovivipare(unsigned int nb_oeufs      ,
                        unsigned int duree_gestation ,
                        bool rarete)
: Vivipare(duree_gestation), // Mauvais ordre !!
  Ovipare(nb_oeufs),
  espece_rare(rarete)
{ }
```

Mais, attention, et c'est là la petite subtilité avec l'héritage multiple : l'ordre d'appel des constructeurs n'est pas celui de l'écriture des constructeurs dans la section « liste d'initialisation » du constructeur de la sous-classe mais les constructeurs sont appelés suivant l'ordre de déclaration de l'héritage. Et, comme toujours, l'ordre d'appel des destructeurs se fait dans l'ordre inverse de l'appel des constructeurs. Illustrons tout ceci sur un exemple concret : reprenons donc notre exemple des « Ovovivipares » qui héritent à la fois d'« Ovipares » et de « Vivipares ». Supposons que donc les « Ovipares » (leur constructeur) pour se construire aient besoin typiquement d'un nombre d'œufs et que les « Vivipares » ont un constructeur qui nécessite une durée de gestation et les « Ovovivipares » on va leur ajouter, disons, un attribut ici qui serait un booléen pour indiquer si l'espèce est rare ou non. Donc, on passerait en troisième paramètre ici un argument « rarete » qui permettrait d'initialiser donc le fait que l'espèce est rare ou non. Donc, typiquement, le prototype du constructeur d'« Ovovivipare » serait d'avoir un paramètre pour le constructeur d'« Ovipare », un paramètre pour le constructeur de « Vivipare » et un paramètre pour pouvoir initialiser son propre attribut et puis on lui donne une valeur par défaut qui est un « false ». Le constructeur d'« Ovovivipare » a donc la charge d'appeler les constructeurs des classes dont il hérite. Donc, premier point, il appellerait dans sa section « liste d'initialisation » les constructeurs des classes dont il hérite, donc par exemple le constructeur de la classe « Vivipare » auquel il passe le paramètre attendu, le constructeur de la classe « Ovipare » auquel il passe le paramètre correspondant, séparés comme d'habitude par

notes

résumé

2m 5s



```
class Ovovivipare : public Ovipare, public Vivipare {
public:
    Ovovivipare(unsigned int nb_oeufs      ,
                 unsigned int duree_gestation ,
                 bool rarete                = false );
    virtual ~Ovovivipare();
protected:
    bool espece_rare;
};

Ovovivipare::Ovovivipare(unsigned int nb_oeufs      ,
                         unsigned int duree_gestation ,
                         bool rarete)
: Vivipare(duree_gestation), // Mauvais ordre !!
  Ovipare(nb_oeufs),
  espece_rare(rarete)
{ }
```

des virgules, et puis il initialise donc son attribut avec le paramètre qu'il a reçu ici, en troisième argument. Simplement ici, deuxième point, il faut faire attention : même si l'on écrit le constructeur d'« Ovovivipare » comme on a l'impression, appelant d'abord le constructeur de « Vivipare » puis le constructeur d'« Ovipare », en fait lorsqu'on envoie appeler le constructeur d'« Ovovivipare », le premier constructeur qui va être appelé c'est le constructeur d'« Ovipare ». Ce qui compte dans l'ordre d'appel des constructeurs des sous-classes dans le cas d'héritage multiple, c'est bien la déclaration d'héritage et non pas l'ordre dans lequel on les a déclarés dans la section liste d'initialisation du constructeur de la sous-classe. Donc, ici, quoi qu'on ait écrit, c'est d'abord le constructeur d'« Ovipare » qui va être appelé et ensuite le constructeur de « Vivipare » qui va être appelé. D'ailleurs, la plupart des compilateurs modernes maintenant indiquent ici une erreur sur l'ordre d'appel des constructeurs et signalent avec un warning, plutôt qu'une erreur, donc un message d'alerte qui vous avertit que, en fait, l'ordre d'appel des constructeurs sera dans un ordre différent que celui que vous pensez avoir écrit ici. que celui que vous pensez avoir écrit ici.

notes

résumé