

Support de cours

Cours:

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

Vidéo:

W16-01-heritmultintro-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Notion d'héritage multiple. Classes d'un programme. Entité des données. Exemple de jeu. Super-classes directes. Exemple concret. Méthodes virtuelles. Notion de classes abstraites. Sequence video. Héritage multiple. Entité du jeu. Œuvre des solutions polymorphiques. Thème de cette séquence. Super-classes. Méthodes virtuelles pures.



[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 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Cette sequence video a pour but de vous présenter

notes

résumé

0m 1s



Où en est-on ?

Nous connaissons maintenant les aspects essentiels de la POO :

- ▶ Encapsulation et abstraction
 - ▶ regroupement traitements et données :
`class Rectangle { ... };`
 - ▶ séparation entre interface et détails d'implémentation :
`public, protected, private`
- ▶ Héritage : relation est-un
`class Guerrier : public Personnage {...};`
- ▶ Polymorphisme
 - ▶ pouvoir être vu de plusieurs façons, abstraction, généricité
 - ▶ 2 ingrédients nécessaires : pointeurs et méthodes virtuelles
 - ▶ classes abstraites et méthodes virtuelles pures



la notion d'héritage multiple en C++. Avant de commencer sur ce thème, un petit tour d'horizon des connaissances acquises jusqu'ici. Vous connaissez maintenant les trois concepts centraux de l'orientée objet, à savoir : encapsulation et abstraction, héritage et polymorphisme. Vous savez qu'encapsuler et abstraire, c'est regrouper en une seule et même entité des données et des traitements. C'est également séparer l'interface d'utilisation des détails d'implémentation. Vous savez aussi que l'héritage va permettre de mettre en place une relation est-un entre différentes classes d'un programme et vous savez qu'il s'agit là d'un outil qui permet de mettre en œuvre des solutions polymorphiques, au sens du polymorphisme d'inclusion. Vous avez appris enfin que le polymorphisme est un outil très puissant qui va permettre de faire en sorte que l'exécution d'un même code s'adapte automatiquement à différents types de données et vous savez qu'en C++ pour qu'une solution polymorphique puisse être mise en œuvre, il y a deux ingrédients nécessaires :

notes

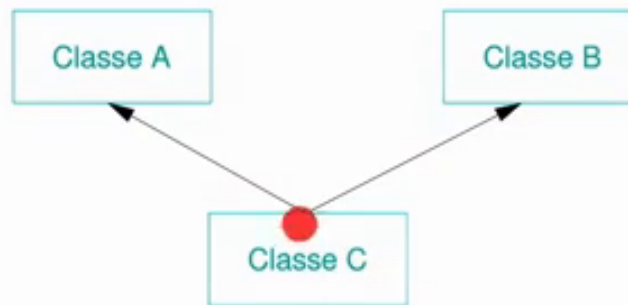
résumé

0m 5s



Qu'est-ce que l'héritage multiple ?

En C++, une sous-classe peut hériter de **plusieurs super-classes** :



Comme pour l'héritage simple, la sous-classe hérite des super-classes :

- ▶ tous leurs *attributs et méthodes* (sauf les constructeurs/destructeurs)
- ▶ leur *type*

on doit utiliser des méthodes virtuelles au travers de références ou de pointeurs. Et pour parachever ce petit tour d'horizon de vos connaissances, mentionnons encore la notion de classes abstraites et de méthodes virtuelles pures qui permettent d'affiner les modèles orientée objet de façon significative. Passons maintenant au thème de cette séquence, l'héritage multiple qui est en fait une continuation d'un thème déjà connu, celui de l'héritage simple. Jusqu'à maintenant, toutes les relations d'héritage que nous avons étudiées étaient des relations d'héritage simple dans le sens où chaque sous-classe n'avait qu'une seule classe parente directe. L'héritage multiple est simplement le fait

notes

résumé

1m 1s



Supposons que l'on souhaite programmer un jeu mettant en scène les entités suivantes :

1. Balle
2. Raquette
3. Filet
4. Joueur

Chaque entité sera principalement dotée d'une méthode `evolve`, gérant l'évolution de l'entité dans le jeu.

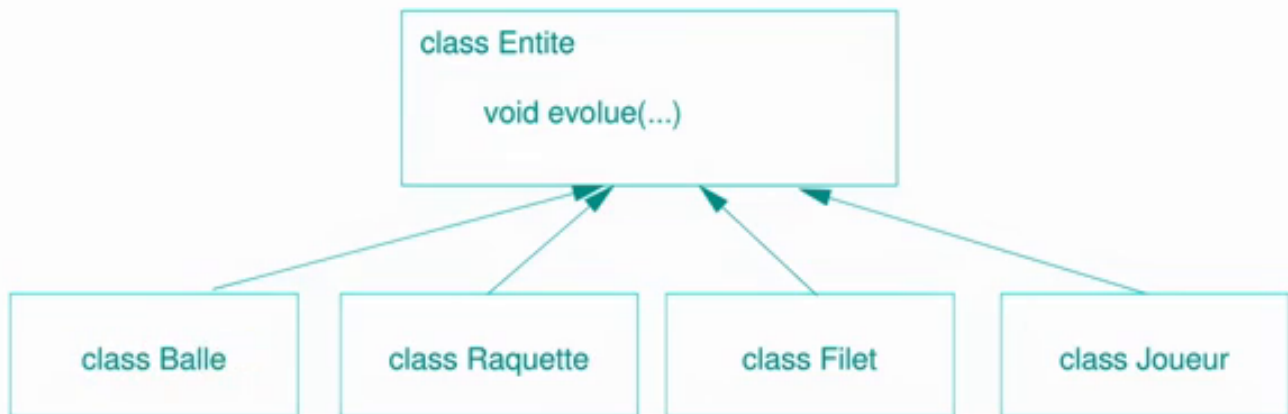
qu'une sous-classe puisse avoir plusieurs super-classes directes. L'héritage multiple est autorisé en C++ : on a le droit de faire en sorte qu'une sous-classe hérite de plusieurs classes parentes directement mais c'est une notion qui n'est pas offerte par tous les langages de programmation orientée objet. Les notions de bases relatives à l'héritage multiple sont tout à fait analogues à celles vues dans le cadre de l'héritage simple, à savoir qu'une sous-classe va hériter de ces super-classes, de tous leurs attributs et méthodes, hormis les constructeurs -destructeurs et de leur type. Voyons sur un exemple concret dans quel contexte l'héritage multiple peut être utile.

notes

résumé

1m 37s





Pour ne pas trop changer nos habitudes, prenons encore un exemple de jeu, un jeu où cette fois différentes entités entrent en scène comme, par exemple, une « Balle », des « Raquettes », un « Filet », un « Joueur ». Chaque entité du jeu sera dotée d'une méthode évolue qui va permettre de gérer l'évolution de l'entité en question dans le jeu. Comme première ébauche de conception, on peut donc imaginer une super-classe « Entite » qui aura pour méthode principale une méthode évolue, très probablement polymorphique et redéfinie dans les sous-classes mais ce n'est pas notre propos ici et des sous-classes représentant les différentes entités concrètes de notre jeu comme

notes

résumé

2m 13s



Si l'on analyse de plus près les besoins du jeu, on réalise que :

- ▶ certaines entités doivent avoir une représentation graphique
(Balle, Raquette, Filet)
- ▶ ... et d'autres non (Joueur)
- ▶ certaines entités doivent être interactives
(on veut par exemple pouvoir les contrôler avec la souris) :
Balle, Raquette
- ▶ ... et d'autres non : Joueur ● Filet

🗨 Comment organiser tout cela ?

la classe « Balle », la classe « Raquette », la classe « Filet » et la classe « Joueur ». Imaginons maintenant qu'une analyse un peu plus fine des besoins du jeu nous fasse réaliser que toutes les entités ne se comportent pas exactement de la même façon, n'ont pas les mêmes besoins. Par exemple, certaines entités doivent pouvoir être affichées, dessinées, avoir une représentation graphique comme, par exemple, la « Balle », la « Raquette », le « Filet » et d'autres non : on imagine par exemple que l'on ne voit pas le « Joueur » dans ce jeu. On peut imaginer aussi que certaines entités doivent être interactives, c'est-à-dire contrôlables avec la souris ou le clavier comme, par exemple, la « Balle » ou la « Raquette » et d'autres non : on ne peut pas contrôler le « Joueur »

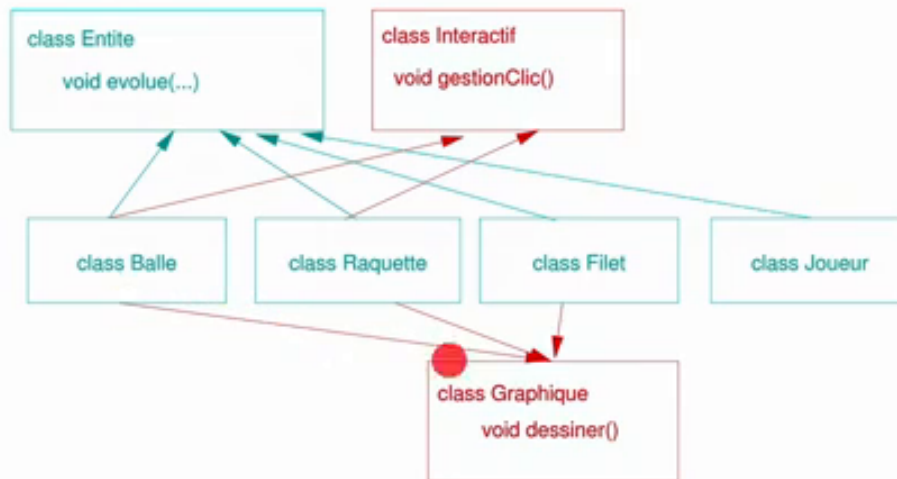
notes


résumé

2m 49s



Il nous faudrait mettre en place une hiérarchie de classes telle que celle-ci :



 Possible en C++ grâce à l'héritage multiple !

ni le « Filet », par exemple. Et on se pose la question de comment organiser tout ceci. La hiérarchie de classe que vous avez sous les yeux ici répond précisément aux besoins que nous venons d'énoncer. Ici, une super-classe « Interactive » a été introduite qui va répondre aux besoins spécifiques des différentes entités qui ont besoin d'être gérées par le biais d'une interaction comme, par exemple, la balle ou la raquette. Cette super-classe va fournir les méthodes qui permettent justement de gérer cette interaction. De façon analogue, la super-classe « Graphique » va permettre de définir toutes les entités que l'on veut dessinables dans le jeu, à savoir la raquette, la balle ou encore le filet.

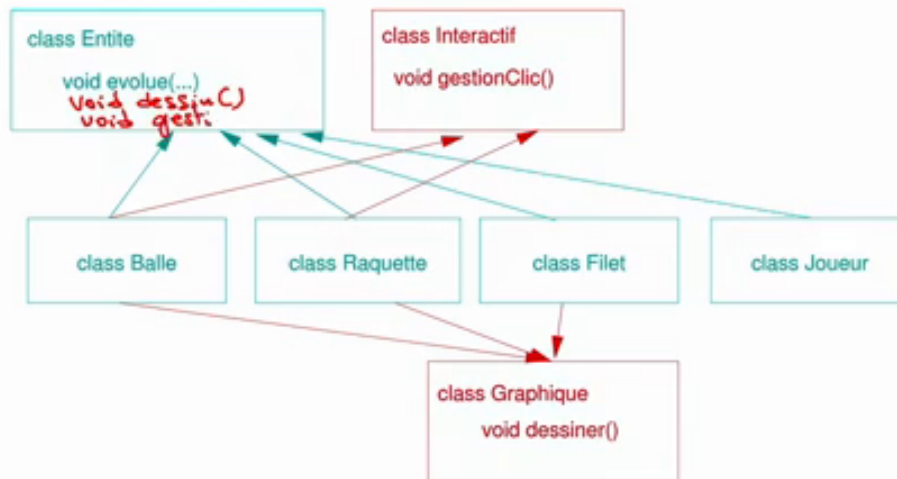
notes

résumé

3m 25s



Il nous faudrait mettre en place une hiérarchie de classes telle que celle-ci :



🎮 Possible en C++ grâce à l'héritage multiple !

C'est cette super-classe qui va fournir les éléments nécessaires à la représentation graphique de ces entités. Ces deux super-classes permettent donc de voir certaines entités, mais pas toutes, comme des objets interactifs et d'autres comme des objets graphiques. Notez qu'il n'aurait pas été correct de placer les méthodes de dessin et de gestion de l'interaction dans la super-classe « Entite ». Pourquoi, à votre avis ? Pourquoi, à votre avis ?

notes

résumé

4m 1s

