

Support de cours

Cours:

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

Vidéo:

W14-02-2-masquage-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Exemple d'une hiérarchie de personnages. Super classe. Implémentations possibles de la méthode. Nom d'attribut. Objet de type. Plupart des classes. Façon satisfaisante. Séquence vidéo précédente. Hiérarchie de classes de la façon. Objet de type guerrier. Cas de notre exemple. Besoins d'une sous-classe. Genre de situations. Cadre d'une hiérarchie de classes. Méthode.



[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 : masquage

(Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Dans certaines situations, des membres tels que définis dans une super classe

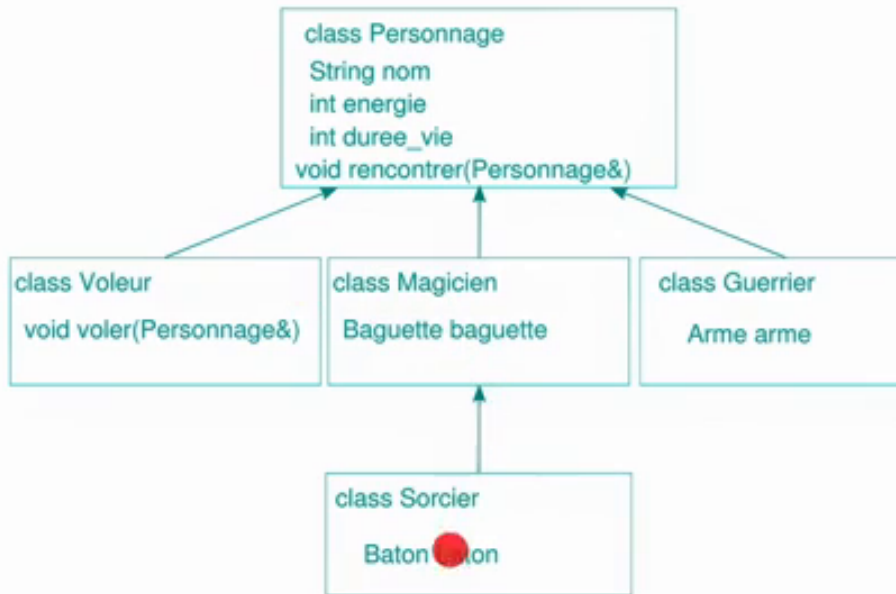
notes

résumé

0m 1s



Exemple : héritage



ne répondent pas de façon satisfaisante aux besoins d'une sous-classe. La redéfinition ou le masquage sont des concepts qui permettent de faire face à ce genre de situations. Dans la séquence vidéo précédente, nous avons pris l'exemple d'une hiérarchie de personnages dans un jeu et nous étions partis de l'hypothèse que tous ces personnages avaient une façon commune de rencontrer un autre personnage. Tout personnage, lorsqu'il en rencontre un autre, va par exemple le saluer. Supposons maintenant que cette façon d'implémenter la méthode « rencontrer »

notes

résumé

0m 5s



- Pour un personnage non-Guerrier :

```
void rencontrer(Personnage& le_perso) const { saluer(le_perso); }
```

- Pour un Guerrier

```
void rencontrer(Personnage& le_pauvre) const { frapper(le_pauvre); }
```

Faut-il re-concevoir toute la hiérarchie ?

- ❏ Non, on ajoute simplement une méthode `rencontrer(Personnage&)` spéciale dans la sous-classe `Guerrier`

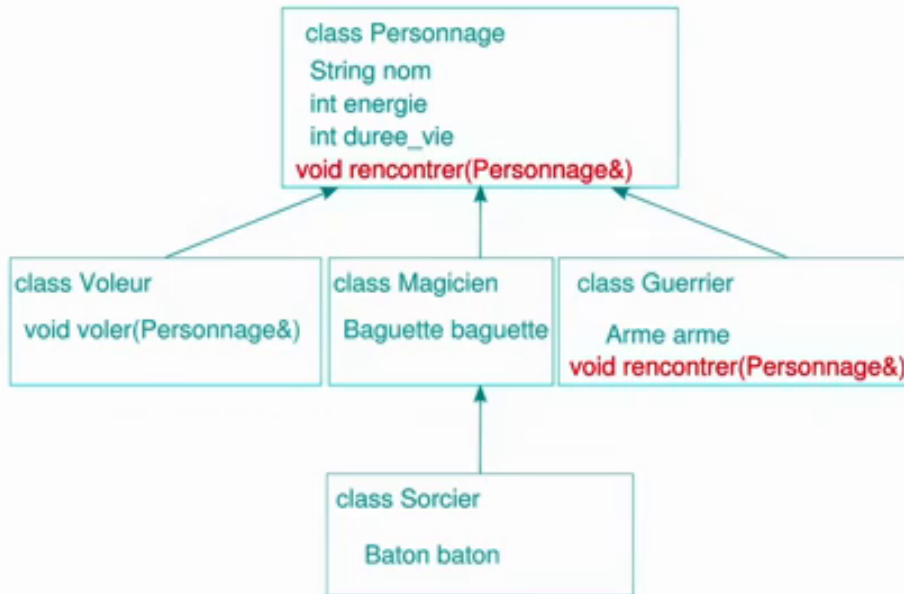
soit satisfaisante pour la plupart des classes, mais pas pour toutes. Imaginons par exemple que le guerrier soit un personnage un peu plus belliqueux que les autres, et lorsqu'il en rencontre un autre, eh bien au lieu de le saluer il le frappe, par exemple. Nous sommes donc dans la situation où nous envisageons deux implémentations possibles de la méthode « rencontrer » : l'une pour les personnages non guerrier, un personnage non guerrier, lorsqu'il en rencontre un autre, va le saluer, et l'autre pour un personnage guerrier, qui, cette fois, lorsqu'il rencontre un autre personnage, eh bien, va le frapper. Qu'en est-il alors de notre première conception ? Faut-il alors reconcevoir toute la hiérarchie ?

notes

résumé

0m 37s





La réponse est heureusement non. Il suffira en fait de spécialiser la méthode « rencontrer » dans la sous-classe « guerrier ». Nous allons donc revisiter notre hiérarchie de classes de la façon suivante : nous allons garder notre méthode « rencontrer » générale qui est satisfaisante pour la plupart des classes dans la super classe « personnage », mais dans la classe « guerrier », nous allons spécialiser cette méthode en lui donnant une définition qui est plus satisfaisante pour la sous-classe en question. Dans le cas de notre exemple, on imagine que la classe « guerrier » serait complétée ici par une méthode « frapper », qui serait invoquée par la méthode « rencontrer ». La méthode « rencontrer » va donc être désormais présente à deux niveaux de la hiérarchie : une fois dans la super classe « personnage » et une autre fois dans la sous-classe « guerrier ». On parle de « masquage », parce que désormais, pour un objet de type « guerrier », c'est la méthode spécialisée qui va primer. On dit qu'elle va « masquer » la méthode plus générale, qui s'applique à défaut d'autres méthodes plus spécialisées. Par exemple, si dans un programme je travaille avec un objet de type « magicien »,

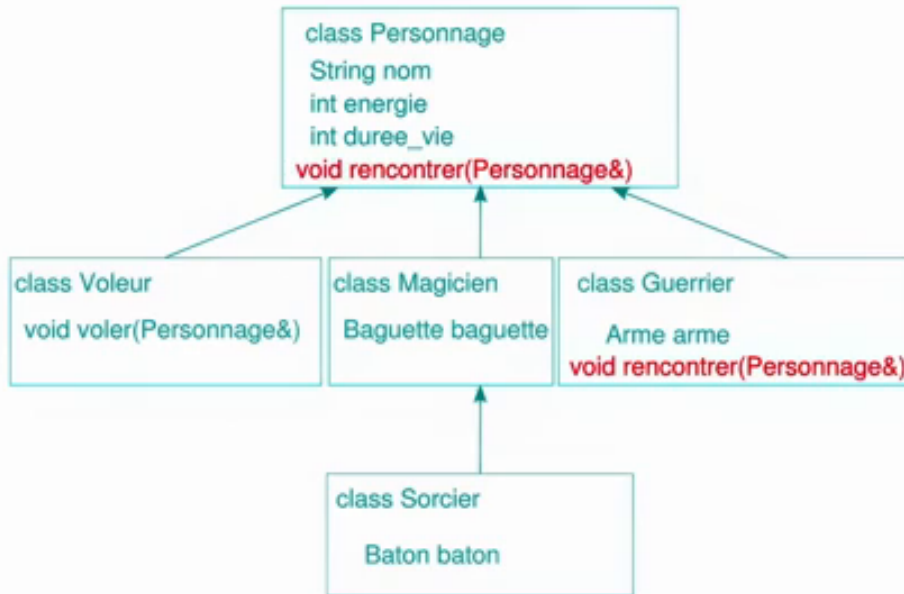
notes

résumé

1m 13s



Les Guerrier font bande à part : masquage



Magicien m (...);
m. rencontrer (...);

Guerrier g (...);
g. rencontre

et que je lui applique la méthode « rencontrer », comme il n'y a pas de définition spécifique de la méthode « rencontrer » dans « Magicien », on va utiliser la méthode héritée, à savoir la méthode « rencontrer » de « Personnage ». Donc ici, cette méthode « rencontrer » sera la méthode générale. Par contre, si maintenant je veux appliquer la méthode « rencontrer » à un objet de type Guerrier,

notes

résumé

2m 25s



- ▶ Masquage : un identificateur qui en cache un autre
- ▶ Situations possibles dans une hiérarchie :
 - ▶ Même nom d'attribut ou de méthode utilisé sur plusieurs niveaux
 - ▶ Peu courant pour les attributs
 - ▶ Très courant et **pratique** pour les méthodes

eh bien, comme il existe dans la classe « Guerrier » une méthode « rencontrer » spécialisée, c'est cette méthode qui ici va être appelée. Donc on va dire que cette méthode va masquer la méthode héritée plus générale.

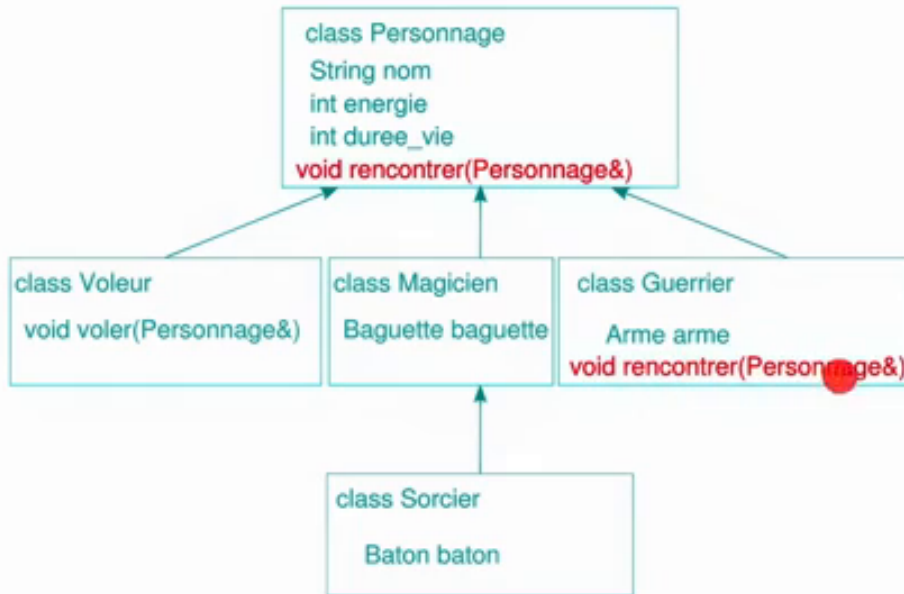
notes

résumé

3m 1s



Les Guerrier font bande à part : masquage



Magicien m (...);
m. rencontrer (...);

Guerrier g (...);
g. rencontrer (...);

En programmation, de façon générale, on parle de « masquage » lorsqu'un identificateur en cache un autre. Dans le cadre d'une hiérarchie de classes, on parlera de « masquage » lorsque le même nom d'attribut ou de méthode est utilisé sur plusieurs niveaux de la hiérarchie. Dans le cadre de notre exemple,

notes

résumé

3m 14s



Masquage dans une hiérarchie



- ▶ **Masquage** : un identificateur qui en cache un autre
- ▶ Situations possibles dans une hiérarchie :
 - ▶ Même **nom d'attribut ou de méthode** utilisé sur plusieurs niveaux
 - ▶ Peu courant pour les attributs
 - ▶ Très courant et **pratique** pour les méthodes

le même en-tête de méthode se retrouvait à deux niveaux de la hiérarchie. Le masquage est peu courant et peu recommandé pour les attributs. Il est source de confusion. Ce qu'il signifie concrètement, c'est la chose suivante : donc, nous aurions une super classe A, par exemple dont hérite une sous-classe B. Avoir un masquage pour les attributs signifie que, par exemple dans la classe A, il y aurait un attribut se nommant « a » et que dans la classe B, également, on déclarerait un attribut « a ». Si l'on utilise l'attribut « a » à l'intérieur d'une méthode de la classe B, eh bien, c'est cet attribut « a » qui va être utilisé. On dit que cet attribut « masque » celui qui est hérité de plus haut. Il n'en demeure pas moins qu'un objet de type B dispose désormais, avec cette façon de faire, de deux attributs : l'un qui lui est spécifique, qui s'appelle « a », et l'un qui est hérité de A, qui s'appelle aussi « a », d'où la source de confusion. Donc, le fait qu'on ait le même type ici ne change rien, il suffit en fait que nous ayons le même nom d'attribut pour avoir une situation de masquage. Et évidemment, dans le cas des attributs : exemple à ne pas suivre. Par contre, comme nous l'avons vu dans l'exemple précédent avec le masquage de la méthode « rencontrer » dans la sous-classe « Guerrier », eh bien, le masquage de méthode est très courant et très pratique : il permet d'adapter une hiérarchie de classes à des situations spécifiques.

notes

résumé

3m 37s

