

Support de cours

Cours:

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

Vidéo:

W17-02-affichage-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Méthode virtuelle. Méthode polymorphique virtuelle. Affichage d'un produit. Opérateur d'affichage. Méthode polymorphique. Instances de cette classe de façon. Comportement polymorphique. Fonctionnalité tel. Fonction externe. Défaut de la méthode d'affichage. Objets de la classe produit. Programmation de l'opérateur d'affichage. Monde extérieur. Méthode. Fonctionnalité de la classe.



[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/>

Etude de cas : affichage polymorphique

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Affichage polymorphique

- Tous les produits sont « affichables », chacun à sa façon

☞ affichage polymorphique ?

Le plus simple est de faire appel à une méthode polymorphique définie au niveau de la super-classe :

```
class Produit {
public:
    virtual void afficher(ostream& sortie) const ;
};

// -----
ostream& operator<<(ostream& sortie, Produit const& machin) {
    machin.afficher(sortie);
    return sortie;
}
```

La réponse est effectivement oui. Bien sûr, ce n'est pas l'opérateur en lui-même qui peut être déclaré comme une méthode virtuelle. Il ne s'agit pas d'une méthode, il s'agit d'une fonction externe à toute classe, donc l'opérateur en lui-même ne peut pas être virtuel, par contre, rien ne l'empêche d'invoquer une méthode polymorphique virtuelle sur l'opérande qu'il doit afficher. L'idée est donc ici de faire en sorte que notre opérateur d'affichage fasse appel à une méthode polymorphique qui serait définie au niveau de la super-classe. Donc concrètement, ici, chez nous, dans la super-classe `Produit`. Notre opérateur permettant l'affichage d'un produit, appellerait donc sur ce produit une méthode `afficher` définie dans la super-classe `Produit` ; bien-sûr cette méthode `afficher` doit être déclarée comme virtuelle pour permettre un comportement polymorphique. Pour rappel, ici, c'est le fait que la méthode soit virtuelle et qu'elle s'applique à une référence vers un produit qui va permettre le comportement polymorphique souhaité. Vous remarquerez que la méthode `afficher` est déclarée comme publique, d'abord, parce que ça peut avoir du sens d'offrir cette fonctionnalité tel quel, au monde extérieur, pour des objets de la classe `Produit`; mais aussi parce que ça va nous permettre de nous affranchir de certaines contraintes pour la programmation de l'opérateur d'affichage.

Typiquement, ici, comme `afficher` est une méthode publiquement accessible, l'opérateur peut être programmé en l'utilisant directement et donc, il n'est pas nécessaire d'avoir recours au friend pour accéder à cette fonctionnalité de la classe produit. Dès l'instant où une méthode, dans une classe est déclarée comme virtuelle cela signifie que, potentiellement, nous allons utiliser des instances de cette classe de façon polymorphique.

notes

résumé

0m 1s



Précisons encore le comportement par défaut :

```
virtual void Produit::afficher(ostream& sortie) const {  
    sortie << valeur;  
}
```

Et donc, il faut penser à les détruire également de façon appropriée en introduisant un destructeur, lui-même virtuel. Nous souhaitons que le comportement par défaut de la méthode d'affichage d'un produit permette d'en afficher le prix. On pourrait imaginer d'écrire la méthode afficher de sorte qu'elle affiche la valeur du produit. Cette façon de procéder n'est cependant pas bonne, sauriez-vous dire pourquoi ? sauriez-vous dire pourquoi ?

notes

résumé

1m 37s

