

Support de cours

Cours:

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

Vidéo:

W15-03-polymasq-CPP-pt5

Concepts (extraits des sous-titres générés automatiquement) :

Classe a.. Concepteur de la sous-classe b. Niveau de votre pratique. Erreur de conception. Mot-clé final. Répétition du mot-clé virtual. Troisième cas. Concepteur de la classe a. Prototype complet de la méthode. Erreurs possibles. Override de la méthode. Premier temps. Niveau de ce cours. Nouvelle méthode. Message du compilateur.



[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/>

Masquage, substitution et surcharge

(Partie 5)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



```

class A {
    // ...
    virtual void f1();
    virtual void f2() const;
        void f3(); // non virtuelle (oubli?)
    virtual void f4() final // pas de redéfinition
};

class B : public A {
    // ...
    virtual void f1() override; // OK
    virtual void f1() override; // Erreur faute de frappe : 1 <-> l
    virtual void f2() override; // Erreur: a oublié le const
        void f3() override; // Erreur: non virtuelle
    virtual void f4(); // Erreur : f4 était final
};

```

L'erreur ici est que la méthode f2 n'est pas const alors qu'elle était définie comme const dans la classe A. Donc la méthode f2 ici est une autre méthode f2. Attention ! const fait vraiment partie de la spécificité de ce qu'on appelle l'en-tête de la méthode, le prototype complet de la méthode contient const et donc ici c'est une autre méthode f2. Donc le compilateur n'est pas d'accord. Ce n'est pas un override de la méthode f2 mais c'est en effet une nouvelle méthode f2. Voilà donc un autre exemple de protection contre les bêtises des développeurs. Oublie du mot const, Faute de frappe. En ce qui concerne donc la méthode f3, peut-être que c'est une erreur de conception au niveau de A. Peut-être que f3 aurait dû être virtuelle et on pensait que c'est une substitution de cette méthode f3. Je vous rappelle que la répétition du mot-clé virtual n'est pas nécessaire. On pourrait tout à fait, même pour une méthode virtuelle, écrire ceci. Le concepteur de la sous-classe B pense redéfinir une méthode virtuelle, donc il écrit override. Là, à nouveau, il a eu un message du compilateur qui lui dit : "Non, non, attention tu ne fais pas ce que tu crois faire". Cette méthode f3 ne peut pas être substituée parce que ce n'est pas une méthode virtuelle. On a ici un pur masquage et sans substitution. Voilà un troisième cas qui permet de prémunir le programmeur de la sous-classe B contre des erreurs possibles. Ici, une erreur qui viendrait soit de son interprétation de ce qu'est la classe A soit carrément du concepteur de la classe A qui a peut-être oublié le mot-clé virtual. Enfin, dernière situation avec le mot-clé final. Ici le concepteur de la classe A empêche toute redéfinition future,

notes

résumé

0m 1s



```
class A {  
    // ...  
    virtual void f1();  
    virtual void f2() const;  
        void f3(); // non virtuelle (oubli?)  
    virtual void f4() final; // pas de redéfinition  
};  
  
class B : public A {  
    // ...  
    virtual void f1() override; // OK  
    virtual void f1() override; // Erreur faute de frappe : 1 <-> l  
    virtual void f2() override; // Erreur: a oublié le const  
        void f3() override; // Erreur: non virtuelle  
    virtual void f4(); // Erreur : f4 était final  
};
```

toute substitution de cette méthode f4. Donc le concepteur de la classe B sous-classe de A n'a pas le droit de redéfinir la méthode f4. Elle est final ici, ça veut dire que l'on n'a pas le droit de redéfinir

notes

résumé



dans les sous-classes cette méthode f4. Donc le compilateur refuserait de compiler cette ligne. Voilà donc pour cet exemple.

notes

résumé

2m 13s



Conseils :

- ▶ `override` : utilisez-le (pour vous prémunir), même si c'est un peu verbeux
- ▶ `final` : oubliez (pour les méthodes)

Note : le mot clé `final` peut aussi s'utiliser pour les classes elles-mêmes pour empêcher la dérivation (interdire les sous-classes) :

```
class Sterile final { ... };  
class C : public Sterile // INTERDIT !
```

Au niveau de votre pratique ce que nous vous conseillons c'est d'utiliser `override`. Même si c'est un petit peu fastidieux de l'écrire. Utilisez-le pour vous prémunir dans un premier temps, du genre d'erreur que l'on a pu vous montrer dans l'exemple précédent. Au niveau de `final`, je pense que ce n'est pas nécessaire au niveau de ce cours, Vous pouvez oublier. A mon avis, ça ne sert absolument à rien. Pourquoi diable voudrait-on empêcher quelqu'un de redéfinir ces méthodes ? Enfin pour être tout à fait complet, je voudrais vous signaler que `final` existe aussi comme mot-clé pour empêcher des sous-dérivations de classes. C'est à dire, empêcher une classe d'avoir des sous-classes. Par exemple, on pourrait ici définir une classe stérile, qui ne peut pas avoir de sous-classe. En rajoutant le mot-clé `final` ici. Donc de cette classe stérile, on n'aurait pas le droit d'hériter.

notes

résumé

2m 20s





notes

résumé

3m 13s

