

Support de cours

Cours:

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

Vidéo:

W15-02-polymvirt-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Méthode virtuelle. Instance réelle. Résolution dynamique des liens. Cas précédent. Biais du pointeur. Classe dauphin. Sous-classe dauphin. Ligne suivante. Affichage suivant. Méthode. Super-classe. Pointeurs. Conséquent. Manger. Passage.



[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/>

Polymorphisme : méthodes virtuelles

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



virtuelle / non virtuelle : un exemple (2)

Que produit le code suivant ?

```
int main() {
    Mammifere* lui(new Dauphin());
    lui->avancer();
    lui->manger();
    delete lui;
    return 0;
}
```

Un nouveau mammifère est né !
Coui, Couic !
Je nage.

Mammifere::Mammifere()
Dauphin::Dauphin()
Dauphin::avancer()

Notez ici au passage que par le fait que nous utilisons des pointeurs, nous permettons à la méthode de s'appliquer à l'instance réelle pointée par l'objet, et donc d'avoir du polymorphisme. Rappelons que la notation « lui->avancer() » est équivalente à la notation « (*lui).avancer() ». Nous accédons au contenu pointé par le pointeur « lui », et à cet objet nous appliquons la méthode « avancer ». La méthode « avancer » est définie comme virtuelle dans la super-classe puis redéfinie dans la sous-classe Dauphin. Comme nous permettons à cette méthode virtuelle d'agir directement sur l'instance réelle par le biais du pointeur, la résolution dynamique des liens a lieu et c'est la méthode « avancer » du Dauphin qui s'applique. Par conséquent, c'est le message « Je nage » qui va s'afficher ici, et nous obtenons à ce stade l'affichage suivant. Passons à la ligne suivante où la méthode « manger » est invoquée

notes

résumé

0m 1s



```

#include <iostream>
using namespace std;

class Mammifere {
public:
    Mammifere() { cout << "Un nouveau mammifère est né !" << endl; }
    virtual ~Mammifere() { cout << "Un mammifère est en train de mourir :(" << endl; }
    ✗ void manger() const { cout << "Miam... croumf !" << endl; }
    virtual void avancer() const { cout << "Un grand pas pour l'humanité." << endl; }
};

class Dauphin : public Mammifere {
public:
    Dauphin () { cout << "Coui, Couic !" << endl; }
    ~Dauphin() { cout << "Flipper, c'est fini..." << endl; }
    void manger() const { cout << "Sglups, un poisson." << endl; }
    void avancer() const { cout << "Je nage." << endl; }
};

```

sur l'objet pointé par « lui ». La méthode « manger » est bien redéfinie dans la classe Dauphin mais attention, contrairement au cas précédent, la méthode manger n'est pas déclarée comme virtuelle dans la super-classe. dans la super-classe.

notes

résumé

1m 13s

