



Support de cours

Cours:

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

Vidéo:

W15-04-polymabstr-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Méthode virtuelle pure. Méthode abstraite. Méthode virtuelle. Notion de périmètre. Niveau de la classe. Figures fermées. Appel de la méthode surface. Méthode de l'instance. Classe. Stricte nécessité. Méthode volume. Fin de son prototype. Travers d'un pointeur. Classe abstraite. Instances de cette 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/>

Classes abstraites

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Méthodes virtuelles pures : définition et syntaxe

Une méthode *virtuelle pure*, ou *abstraite* :

- ▶ sert à imposer aux sous-classes (non abstraites) qu'elles **doivent redéfinir** la méthode virtuelle héritée
- ▶ est signalée par un `= 0` en fin de prototype,
- ▶ est, en général, *incomplètement spécifiée* : il n'y a très souvent *pas de définition* dans la classe où elle est introduite (pas de corps).

Syntaxe :

virtual Type nom_methode(liste de paramètres) = 0;

Exemple :

```
class Personnage {
    // ...
    virtual void afficher() const = 0;
    // ...
};
```

Voyons maintenant comment tout ceci s'écrit concrètement en C++. Une méthode virtuelle pure, on appelle aussi cela une méthode abstraite, sert donc à imposer aux autres sous-classes qu'elles doivent redéfinir la méthode virtuelle héritée. Pour définir une méthode virtuelle pure, une méthode abstraite, on rajoute simplement « = 0 » à la fin de son prototype. Par exemple, si dans la classe « Personnage », on aurait écrit que c'est une méthode virtuelle « afficher », la méthode « afficher » ne modifie pas un personnage, et à la fin de son prototype, au lieu d'avoir un point virgule, on rajoute « = 0 ». Ceci définit donc une méthode virtuelle pure, une méthode abstraite.

notes

résumé

0m 1s



```
class FigureFermee {  
public:  
    virtual double surface() const = 0;  
    virtual double perimetre() const = 0;  
  
    // On peut utiliser une méthode virtuelle pure :  
    double volume (double hauteur) const {  
        return hauteur * surface();  
    }  
};
```

En général, une méthode virtuelle pure est incomplètement spécifiée, c'est-à-dire qu'il n'y a souvent pas de définition, mais juste un prototype dans la classe où elle est introduite. Ceci n'est pas une stricte nécessité mais c'est souvent le cas, on ne définirait pas par exemple ici la méthode « afficher » pour un personnage générique, on attend que les sous-classes définissent leur propre méthode « afficher » spécifique aux guerriers, magiciens, aux voleurs... Si je reprends l'autre exemple, celui des figures fermées, on aurait donc une méthode « surface », qui serait une méthode virtuelle pure, donc ici on rajoute « = 0 » derrière son prototype, que l'on pourrait quand même utiliser, on a le droit d'utiliser une méthode virtuelle pure, on peut tout à fait définir une méthode volume, qu'on n'a pas définie comme virtuelle, il n'y a aucune raison qu'une sous-classe redéfinisse cette méthode volume, qui prend donc une hauteur, et qui calcule le volume engendré, comme le produit de la hauteur, par l'appel de la méthode surface,

notes

résumé

0m 49s





laquelle méthode surface sera la méthode de l'instance qui sera appelée.

notes

résumé

1m 49s



```
class FigureFermee {
public:
    virtual double surface() const = 0;
    virtual double perimetre() const = 0;

    // On peut utiliser une méthode virtuelle pure :
    double volume (double hauteur) const {
        return hauteur * surface();
    }
};
```

Car en effet, comme nous allons le voir, une classe qui contient une méthode virtuelle pure, comme « FigureFermee », ne peut pas être instanciée, et donc de ce fait, on n'appellera jamais la méthode surface directement sur une figure fermée. Par contre on peut imaginer des sous-classes, comme Rectangle, Cercle, ... qui savent calculer leur surface.

notes

résumé

1m 52s



```
class FigureFermee {
public:
    virtual double surface() const = 0;
    virtual double perimetre() const = 0;

    // On peut utiliser une méthode virtuelle pure :
    double volume (double hauteur) const {
        return hauteur * surface();
    }
};
```

th → surface()

Donc on appellera la méthode de « Cercle » s'il s'agit d'un cercle, on appellera la méthode de « Rectangle » s'il s'agit d'un rectangle, ça c'est l'aspect polymorphique, par ce mot clé « virtual » qui pourra prendre place. On a bien ici appel de la méthode surface au travers d'un pointeur, car je vous rappelle que ceci est exactement la même chose que « this - - > surface », on a parfois tendance à l'oublier, et donc on a bien ici le polymorphisme qui peut prendre place

notes

résumé

2m 8s



```
class FigureFermee {  
public:  
    virtual double surface() const = 0;  
    virtual double perimetre() const = 0;  
  
    // On peut utiliser une méthode virtuelle pure :  
    double volume (double hauteur) const {  
        return hauteur * surface();  
    }  
};
```

this → surface()

puisque on a bien appel au travers d'un pointeur, même s'il n'est pas explicitement écrit, et un aspect virtuel. Une autre méthode virtuelle pure que l'on pourrait définir

notes

résumé

2m 37s




```
class FigureFermee {  
    public:  
        virtual double surface() const = 0;  
        virtual double perimetre() const = 0;  
  
    // On peut utiliser une méthode virtuelle pure :  
    double volume (double hauteur) const {  
        return hauteur * surface();  
    }  
};
```



this → surface()

au niveau de la classe « FigureFermee », c'est la notion de périmètre qui calculerait la longueur d'une figure fermée, le périmètre, et on ne saurait pas non plus a priori le définir de façon générale au niveau de la classe très générique, abstraite, « FigureFermee », mais on pourrait imaginer pouvoir le définir proprement dans des sous-classes. Nous allons reprendre cet exemple dans un instant.

notes

résumé

2m 47s



Une **classe abstraite** est une classe contenant *au moins une méthode virtuelle pure*.

- ▶ Elle *ne peut être instanciée*
 - ▶ Ses sous-classes *restent abstraites* tant qu'elles ne fournissent pas les définitions de *toutes les méthodes virtuelles pures* dont elles héritent.
- (En toute rigueur : tant qu'elles ne suppriment pas l'aspect virtuel pur (le « = 0 »).)

~~Personne p;~~

Un exemple « concret »...

Une classe, qui comme ceci, contient au moins une méthode virtuelle pure, c'est ce qu'on appelle une « classe abstraite ». C'est en effet une classe abstraite, car si une classe contient une méthode virtuelle pure, alors elle ne peut pas être instanciée, on ne peut pas créer d'instances de cette classe. Par exemple, on ne peut pas déclarer de variable de type « personne », et évidemment toute sous-classe qui hérite d'une super classe abstraite

notes

résumé

3m 11s



Classes abstraites : exemple

Une autre équipe crée la sous-classe `Guerrier` de `Personnage` et veut l'utiliser :

```
Jeu jeu;
jeu.ajouter_personnage(new Guerrier(...));
```

S'ils ont oublié de définir la méthode `afficher`,
le code ci-dessus génère une erreur de compilation
car on ne peut pas créer d'instance de `Guerrier` :

```
cannot allocate an object of abstract type 'Guerrier'
because the following virtual functions are pure within 'Guerrier':
virtual void Guerrier::afficher()
```

Personnage
↑
Guerrier

*virtual void
afficher() const
= 0;*

reste abstraite tant qu'elles ne fournissent pas de définition à la méthode virtuelle pure qu'elles ont hérité. En toute rigueur, c'est tant qu'elles ne suppriment pas « = 0 » dans les prototypes de leur redéfinition de la méthode virtuelle pure héritée de la super classe. Regardons tout ceci concrètement sur un exemple, revenons à notre exemple de jeu avec des personnages, et où un « Guerrier » est un « Personnage », et a hérité une méthode virtuelle pure, une méthode virtuelle pure « afficher » de la classe « Personnage ». Si au niveau de la classe « Guerrier », on ne redéfinit pas la méthode « afficher », alors à ce moment là, la sous-classe « Guerrier » est aussi une classe abstraite,

notes

résumé

3m 37s



Classes abstraites : autre exemple

Figure Fermée
perimetre() ---=0; surface() ---,
Cercle Polygone

EPFL

```
class Cercle: public FigureFermee {
public:
    double surface() const override {
        return M_PI * rayon * rayon;
    }
    double perimetre() const override {
        return 2.0 * M_PI * rayon;
    }
protected:
    double rayon;
};
```

Cercle n'est pas une classe abstraite

Cercle c;

```
class Polygone: public FigureFermee {
public:
    double perimetre() const override {
        double p(0.0);
        for (auto cote : cotes) {
            p += cote;
        }
        return p;
    }
protected:
    vector<double> cotes;
};
```

Polygone reste par contre une classe abstraite



puisqu'elle a hérité une méthode virtuelle pure, qu'elle n'a pas redéfinie, donc qui reste virtuelle pure, et donc si l'on essaie d'ajouter comme ça un « Guerrier », -- une instance de la classe « Guerrier » -- au jeu, on va avoir une erreur du compilateur, qui dit que je ne peux pas créer d'objets, d'instances de la classe abstraite « Guerrier », « abstract type », la classe abstraite « Guerrier », car la méthode virtuelle « afficher » dans « Guerrier » est une méthode virtuelle pure. Complétons par notre second exemple sur les figures fermées, nous avons donc toujours les figures fermées qui ont deux méthodes virtuelles pures, une méthode périmètre et une méthode surface, et imaginons donc qu'on ait une classe « Cercle » qui hérite de figures fermées ainsi qu'une sous-classe « Polygone » qui hérite aussi de figures fermées. Dans la sous-classe « Cercle », nous redéfinissons la méthode surface, la surface d'un cercle c'est π fois le rayon au carré, la classe « Cercle » aurait un attribut spécifique rayon, et on redéfinit aussi le périmètre, comme étant « 2 fois π fois le rayon ». La classe « Cercle » définit donc proprement surface et périmètre. La classe « Cercle » a donc redéfini les deux méthodes virtuelles pures sans qu'elles ne restent virtuelles pures, sans rajouter le « = 0 » ici, donc la classe « Cercle » n'a plus de méthode abstraite, c'est donc une classe dont on peut créer des instances, il n'y a aucun souci à déclarer une instance d'un « Cercle ». De l'autre côté, la classe « Polygone », Admettons qu'un polygone soit défini par un tableau dynamique de côtés. Cette classe « Polygone » aurait une redéfinition de la méthode périmètre, sans qu'elle soit abstraite, donc enlève l'aspect abstrait

notes

résumé

4m 25s



Classes abstraites : autre exemple

Figure Fermée
perimetre() ---, surface() ---,
Cercle Polygone

EPFL

```
class Cercle: public FigureFermee {
public:
    double surface() const override {
        return M_PI * rayon * rayon;
    }
    double perimetre() const override {
        return 2.0 * M_PI * rayon;
    }
protected:
    double rayon;
};
```

Cercle n'est pas une classe abstraite

Cercle c;

```
class Polygone: public FigureFermee {
public:
    double perimetre() const override {
        double p(0.0);
        for (auto cote : cotes) {
            p += cote;
        }
        return p;
    }
protected:
    vector<double> cotes;
};
```

Polygone reste par contre une classe abstraite



à périmètre, et redéfinit bien effectivement cette méthode périmètre, mais par contre la classe « Polygone » ne redéfinit pas la surface, on trouve trop compliqué d'avoir une formule ici, pour redéfinir de façon générale la surface d'un polygone, et donc on ne redéfinit pas la méthode surface dans la classe « Polygone ». Donc « Polygone » reste donc de ce fait une classe abstraite, on ne peut pas déclarer d'instances de « Polygone », ce n'est pas possible parce que dans la classe « Polygone », il reste une méthode virtuelle pure qui est la méthode surface héritée de figures fermées, et comme la classe « Polygone » a encore une méthode virtuelle pure, c'est une classe abstraite, on ne peut donc pas en créer d'instances. on ne peut donc pas en créer d'instances.

notes

résumé