Classes abstraites [ka]
Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

C++ - Classes abstraites

Mots-Clés Classes abstraites ■
Requis Classes, Relations entre classes, Héritage, Polymorphisme d'inclusion ■
Difficulté ●●○ (30 min) ■

Introduction

L'héritage est une technique de conception si puissance que certaines classes ne seront pas conçues pour être instanciées mais uniquement pour faciliter l'utilisation et la cohérence du système grâce à l'héritage. Ce module présente les **méthodes** et les **classes abstraites**.

1 Méthode abstraite

1.1 Méthode abstraite

Au sommet d'une hiérarchie de classe, il n'est pas toujours possible de donner une définition générale de certaines méthodes **compatibles** avec toutes les sous-classes, et ce même si l'on sait que toutes ces sous-classes vont effectivement implémenter ces méthodes.

Exemple

Vous voulez créer des classes représentant des figures géométriques afin de concevoir un logiciel de conception par ordinateur. Dans ces logiciels (2D et 3D), les différentes figures tracées sont toutes leurs propriétés comme leurs dimensions, surface, périmètre, volume, couleur de remplissage, couleur du contour, etc. Les valeurs de ces attributs varient d'une figure à l'autre mais certaines méthodes peuvent être identiques comme celles de la couleur, et d'autres non comme le calcul des surfaces, volumes et périmètres. Pourtant toutes ces figures (carrés, rectangles, polygones, triangles, trapèzes, losanges, etc.) doivent disposer de ces méthodes. Comment décrire ceci ?

La solution consiste à écrire une classe de base qui deviendra la super-classe de toutes les figures géométriques et qui va contenir tous les attributs de base comme la position de départ du tracé de la figure, ses couleurs mais aussi toutes les méthodes non seulement de base mais qui doivent aussi obligatoirement être implémentées dans toutes les classes qui en dérivent. Les méthodes de calcul de surface, de volume, de périmètre doivent être déclarées dans cette classe. Pourtant elles ne contiendront rien comme code : ce seront des méthodes abstraites et ce sera à la classe dérivée de les programmer.

1.2 Déclaration d'une méthode abstraite

Méthode abstraite

Dite aussi **méthode virtuelle pure**, elle est incomplètement spécifiée :

- Elle n'a pas de définition (pas de corps) dans la classe où elle est introduite.
- Elle sert à signaler aux sous-classes qu'elles doivent redéfinir la méthode virtuelle héritée.

C++ : Déclaration d'une méthode abstraite

```
class B {
  virtual T mv(...) [const] = 0;
};
```

Explication

Déclare la méthode `mv` comme étant une méthode abstraite signalée par la conjonction de la déclaration `virtual` et par l'affectation d'un pointeur nul (valeur de l'initialiseur « `=0` » à la fin du prototype).

Remarque

Si une classe dérivée ne contient pas de définition d'une méthode déclarée virtuelle pure dans la classe de base, cette méthode sera à nouveau implicitement virtuelle pure dans la classe dérivée.

2 Classe abstraite

2.1 Déclaration d'une classe abstraite

Classe abstraite
Classe contenant **au moins** une méthode abstraite.

Conséquences

- Elle ne peut pas être instanciée. Il est cependant possible de déclarer des références ou pointeurs sur une classe abstraite.
- Ses sous-classes restent abstraites tant qu'elles ne fournissent pas les définitions de toutes les méthodes abstraites dont elles héritent.
- Toute l'ascendance d'une classe abstraite est abstraite.

Déclaration d'une classe abstraite
Il n'y a pas de mot-clé spécifique.

```
class B {
  virtual T mv(...) [const] = 0;
  ...
};
```

Explication
Déclare `B` comme étant une classe abstraite, c.-à-d. qu'elle contient au moins une méthode abstraite.

2.2 Classe abstraite/concrète

Classe concrète
Classe dérivée qui définit **toutes** les méthodes abstraites dont elle a héritée.

Attention
Créer une classe abstraite dérivée à partir d'une classe concrète est interdit !

Exemple
Prenons l'exemple d'un jeu permettant de faire jouer des joueurs humains contre une intelligence artificielle. Dans votre conception, vous introduisez donc une classe `JoueurHumain` pour pouvoir faire jouer le joueur derrière son ordinateur ou sa console vidéo, et une classe `JoueurMachine` pour faire jouer l'ordinateur. Seulement ces deux classes sont tellement proches, qu'elles seront presque interchangeables : le jeu devrait permettre les parties :

- `JoueurHumain` – `JoueurHumain`
- `JoueurHumain` – `JouerMachine`
- `JouerMachine` – `JouerMachine`

Or il n'y a pas d'héritage entre les classes `JoueurHumain` et `JoueurMachine`. Pour les relier, il faut introduire, pour la clarté et la simplification de la conception, une classe de base `Joueur` comme suit :

FIGURE 1 – fig.6-21

Votre partie sera réalisée soit contre un `JoueurHumain`, soit contre un `JoueurMachine`, mais jamais contre un `Joueur`. Cette classe `Joueur` ne sera pas utilisée directement dans votre programme de jeu, sauf pour le polymorphisme.

Conclusion
Dans quels cas concevoir une classe abstraite ?

- Pour clarifier la conception.
- Pour imposer à des classes les mêmes attributs et la même signature pour certaines méthodes.
- Pour écrire des algorithmes plus génériques grâce au polymorphisme.

Les langages successeurs de C++ proposent également la notion d'**interface**.

2.3 Interface

Interface
Classe abstraite dont **toutes** les méthodes sont *implicitement* abstraites et qui ne peuvent posséder d'attributs, à l'exception de constantes.

C++ : Déclaration d'une classe Interface
Il n'y a pas de mot-clé spécifique.

Explication
Déclare la classe `I` comme étant une interface.

2.4 Implémentation d'une Interface

Implémentation d'une Interface
On dit qu'une classe qui décide d'utiliser une interface implémente les méthodes de l'interface, donc **implémente** l'interface.

C++ : Implémentation d'interfaces
Il n'y a pas de mot-clé spécifique.

Explication
Déclare la classe `B` comme implémentant les interfaces `I1`, `I2`, etc.

Remarque
Il faut implémenter **toutes** les méthodes de l'interface dans la classe. Si vous ne le faites pas, la méthode qui n'est pas implémentée doit être déclarée abstraite et la classe devient abstraite, donc non instanciable.

2.5 Exemples

Exemple : Classe abstraite

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

La classe `FigureFermee` déclare deux méthodes virtuelles pures. Elle n'est pas instanciable.

Exemple : Sous-classe concrète

```
class Cercle : public FigureFermee { ... public:
   virtual double surface() const { return M_PI * r * r; }
   virtual double perimetre() const {return 2 * M_PI * r; }
};
```

La classe `Cercle` définit ses deux méthodes abstraites : elle est donc concrète, c.-à-d. instanciable.

Exemple : Sous-classe abstraite

```
class Rectangle : public FigureFermee { ... public:
   virtual double surface() const { return w * h; }
};
```

La classe `Rectangle` reste abstraite (c.-à-d. non instanciable).