Travaux dirigés C++ n°4

Informatique

—IMAC 2e année—

Héritage et polymorphisme

Le but de ce TD est de comprendre l'intérêt et de mettre en place les mécanismes d'héritage.

Nous voulons représenter des figures géométriques. Commençons par les rectangles. Pour les représenter, nous écrirons, par exemple, la classe Rectangle suivante :

```
class Rectangle {

private:

double largeur;
double hauteur;

public:

Rectangle();
Rectangle(double l, double h);

double getLargeur() const;
double getHauteur() const;

void setLargeur(double l);
void setHauteur(double h);

double surface() const;
};
```

▸ **Exercice 1.**
Dans le fichier ***Rectangle.cpp****, écrivez l'implémentation des méthodes précédente et dans un fichier main.cpp, écrivez la fonction main qui teste votre classe Rectangle avec :*

```
Rectangle r1(2.6, 4.42), r2;
std::cout << r1.surface() << std::endl;
r1.setLargeur(3.9);
```

▸ **Exercice 2.** *On souhaite maintenant représenter des carrés. Écrire la classe* ***Carre*** *en se basant sur le modèle de la classe précédante. Vous aurez une variable membre cote et les méthodes getCote et setCote. Dans la méthode main, ajoutez et testez :*

```
Carre c1(3.8), c2;
std::cout << c1.surface() << std::endl;
c2.setCote(2.9);
```

En comparant la classe Carre à celle de Rectangle, on remarque qu'elle lui ressemble énormément. On peut alors se demander s'il est possible de réutiliser la classe Rectangle pour définir la classe Carre.

La réponse est oui grâce à la notion d'héritage qui permet de définir une classe B, appelée classe dérivée, par spécialisation/extension d'une autre classe A, appelée superclasse (ou classe mère, ou encore classe de base).

Pourquoi est-il naturel de faire hériter la classe Carre de Rectangle ? Tout simplement parce qu'un carré est-un rectangle particulier (la largeur est égale à la longueur).

Une classe dérivée est donc une spécialisation de sa superclasse. Toutefois, elle peut aussi être vue comme une extension dans le mesure où on pourra ajouter de nouveaux attributs et méthodes dans la classe dérivée. D'une façon générale, à chaque fois que la relation est-un peut être appliquée entre deux classes, la relation d'héritage devra être utilisée. La classe dérivée B est héritière au sens qu'elle possède tous les attributs et les méthodes de sa superclasse A en plus de ses propres attributs et méthodes. Toutefois, elle n'hérite pas des constructeurs, du destructeur, du constructeur de copie, ni de l'opérateur d'affectation. L'en-tête de la déclaration une classe B qui dérive d'une classe A, sera :
class B : mode dérivation A
où mode dérivation est soit **private** (comportement par défaut), **protected** ou **public**.

L'héritage est contrôlé par le mode dérivation. S'il est **private**, tous les attributs et les méthodes **public** ou **protected** de la superclasse deviennent privés dans la classe dérivée. Si le mode est **proctected**, ils deviennent protégés. Enfin, si le mode est **public**, tous les attributs et les méthodes de la superclasse gardent leur mode d'accès dans la classe dérivée. Ce dernier mode est le mode de dérivation habituel.

La classe dérivée est donc amenée, si nécessaire, à définir ses propres constructeurs, destructeur, constructeur de copie, et opérateur de copie. Lorsqu'elle définit son constructeur, par défaut le constructeur par défaut de la superclasse est d'abord exécuté. Mais, le constructeur de la classe dérivée peut également faire appel à un constructeur particulier de la superclasse avec la notation : $B(\ldots) : A(\ldots)\{\}$.

Le constructeur de la classe A sera exécuté **avant** celui de B.

Questions

- Quelle est la syntaxe qui permet de mettre en place l'héritage entre deux classes?
- Quelle est la signification du mot clé "**protected**"'?
- Comment est appelé le constructeur de la classe mère?

▸ **Exercice 3.**

Récrivez la classe ***Carre*** *en la faisant hériter de la classe Rectangle selon le mode public. Dans la classe Rectangle, vous déclarerez les variables largeur et hauteur protected, et vous définirez le constructeur Carre(double c) qui initialise le coté du carré. Compilez et exécutez à nouveau votre fonction main. Constatez que dans l'énoncé c1.surface(), la méthode appliquée à un objet de type Carre est obtenue par héritage de la classe Rectangle.*

▸ **Exercice 4.**

Lorsque un objet de la classe dérivée est détruit, le destructeur de cette classe dérivée s'applique d'abord, puis celui de la superclasse. Mettez en évidence ce comportement, en définissant un destructeur dans Rectangle et un autre dans Carre.

▸ **Exercice 5.**

Dans le fichier ***Rectangle.cpp****, donnez l'implémentation de la méthode : std::string quiSuisJe() const; pour qu'elle renvoie la chaîne de caractères "je suis un rectangle".*

▸ **Exercice 6.**

Dans votre fonction main, appliquez la méthode quiSuisJe sur r1 et puis sur c1. Compilez et testez votre méthode main. Que constatez-vous ?

Une classe dérivée hérite d'attributs et méthodes de sa superclasse, elle peut aussi définir ses propres variables et méthodes membres, mais elle peut également redéfinir des méthodes héritées. Elle le fait lorsqu'elle désire modifier l'implémentation d'une méthode d'une classe parent, en particulier pour l'adapter son action a‘ des besoins spécifiques. C'est le cas ici pour la méthode quiSuisJe à redéfinir dans Carre.

▸ **Exercice 7.**

Dans la classe Carre, redéfinissez la méthode quiSuisJe pour qu'elle renvoie la chaîne de caractères "Je suis un carre".

Attention : si la superclasse possède plusieurs méthodes surchargées, la classe dérivée devra redéfinir toutes ces méthodes.

▸ **Exercice 8.**

Recompilez et exécutez votre programme pour obtenir le résultat attendu.