Examen de Programmation Orientée Objet

5 Mai 2020

Master 1 Bio-Informatique Bordeaux

Martin DRANCÉ

Exercice 1:

Pour cet exercice, comme indiqué dans la consigne, les classes sont déclarées dans les fichiers *.hpp*. Pour le bon fonctionnement du *main.cpp*, la définition des méthodes est faite directement dans le fichier *main.cpp*.

Pour tester l'utilisation du programme, se placer dans le répertoire qui contient les fichiers *Personne.hpp*, *Patient.hpp* et *main.hpp*. En utilisant le compilateur g++, tapez dans la console **g++ -c main.cpp** puis **g++ main.o**. Puis pour lancer l'exécutable, tapez dans la console **./a.out**.

Pour que la classe **Patient** puisse hériter de la classe **Personne**, il faut passer les attributs de la classe **Personne** de *private* à *protected*. Ce faisant, on respecte l'encapsulation des attributs de la classe **Personne** mais on les rends accessible aux classes filles. Ensuite, il faut déclarer la méthode *affiche()* comme virtuelle. Définir cette méthode comme virtuelle permet de définir deux comportements différents, un pour une instance de la classe **Personne** et un pour une instance de la classe **Patient**. Enfin, il faut déclarer un destructeur dans la classe **Personne** et dans la classe **Patient**. Ces constructeurs doivent être virtuels eux aussi, un destructeur doit toujours être virtuel si l'on souhaite utiliser le polymorphisme.

Exercice 2:

Pour tester l'utilisation du programme, se placer dans le répertoire qui contient les fichiers Forme*hpp* et *main.hpp*. En utilisant le compilateur g++, tapez dans la console **g++ -c main.cpp** puis **g++ main.o**. Puis pour lancer l'exécutable, tapez dans la console **./a.out**.

Lors de la première exécution du programme, une erreur de compilation apparaît : *main.cpp:5:11: error: cannot declare variable 'F' to be of abstract type 'Forme'*. Cette erreur indique que nous essayons d'instancier un objet à partir d'une classe abstraite. Une classe abstraite est une classe qui contient une méthode virtuelle pure et qui servira de base à d'autres classe héritées. Une méthode virtuelle pure est une méthode qui n'est pas déclarée pour la classe où elle est définie, elle devra cependant être déclarée dans les classes qui héritent de la classe abstraite. Ici, la méthode virtuelle pure est *virtual void affiche() = 0* dans la classe **Forme**.

Il existe deux façons de résoudre ce problème :
- Si nous souhaitons conserver la classe **Forme** comme abstraite, nous laissons la méthode *affiche()* comme virtuelle pure. Nous ne pourrons alors plus créer d'objet de la classe **Forme** dans le main, mais des instances de classes héritant de **Forme** pourront être créées.
-Si nous ne souhaitons pas particulièrement garder la classe **Forme** comme abstraite, nous pouvons retirer la méthode virtuelle pure en la transformant en virtuelle, il suffit de retirer le *=0* à la fin de la déclaration de celle-ci. Il faudra alors la définir et non plus uniquement la déclarer, nous pourrons alors instancier des objets de la classe **Forme**.

Dans les deux cas, le résultat affiché sur le terminal est celui de l'Illustration 1.

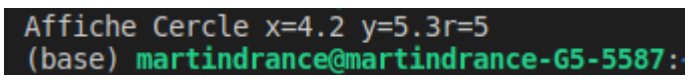

Illustration 1: Affichage sur le terminal après compilation et exécution du fichier main.cpp

Exercice 3:

Comme indiqué dans la consigne, la déclaration et la définition des méthodes sont faites dans le fichier *ListeOrd.hpp*.

Pour tester l'utilisation du programme, se placer dans le répertoire qui contient les fichiers *ListeOrd.hpp* et *main.hpp*. En utilisant le compilateur g++, tapez dans la console **g++ -c main.cpp** puis **g++ main.o**. Puis pour lancer l'exécutable, tapez dans la console **./a.out**.

Après avoir suivit les étapes décrites dans le sujet, nous obtenons sur le terminal ce qui se trouve sur l'Illustration 2.
De base, la liste est remplie avec 3 éléments (8, 2 puis 5). On constate que chacun des éléments est ajouté à sa place dans la liste, le plus grand au début. Ensuite, on retire les deux premiers éléments de la liste. On obtient bien une liste comportant le bon nombre d'éléments.

```
Taille de la liste = 3
[0] : 8
[1] : 5
[2] : 2

Taille de la liste = 1
[0] : 2
(base) martindrance@martindrance-G5-5587:
```

Illustration 2: Affichage sur le terminal après compilation et exécution du fichier main.cpp

La création de deux classes templates **CElement** et **ListeOrd** permet d'appliquer les méthodes de ces classes à n'importe quel objet que l'on voudrait stocker et garder dans un ordre précis.
La méthode *insertInPlace()* de la classe **ListeOrd** est celle qui insérera l'élément au bon endroit dans la liste. Pour ce faire, la première étape est d'insérer le nouvel élément au début de la liste. Puis, à l'aide de deux pointeurs temporaires, on parcourt la liste en regardant si l'élément que l'on vient d'ajouter est inférieur au suivant. Si c'est le cas on échange la valeur de ces deux éléments. Sinon on s'arrête, le nouvel élément est à la bonne place.

Il est à noter que cette méthode de tri se rapproche du *« bubble sort »* dans son cas le plus favorable, puisqu'à chaque ajout d'un élément, la liste de base est déjà triée, ce qui rend la complexité de cet algorithme linéaire.

Exercice 4:

Pour la réalisation de cet exercice, les déclarations et définitions des méthodes de la classe **ListeOrd** sont laissées dans le fichier *ListeOrd.hpp.* Pour les classes **Personne** et **Patient,** comme dans l'exercice 1, les déclarations sont faites dans les fichiers *.hpp* mais les définitions sont laissées dans le fichier *main.cpp*.

Pour tester l'utilisation du programme, se placer dans le répertoire qui contient les fichiers *ListeOrd.hpp, Personne.hpp, Patient.hpp* et *main.hpp*. En utilisant le compilateur g++, tapez dans la console **g++ -c main.cpp**, puis **g++ main.o**. Puis pour lancer l'exécutable, tapez dans la console **./a.out**.

Après avoir suivit les étapes décrites dans le sujet, nous obtenons sur le terminal ce qui se trouve sur l'Illustration 3. On constate que chaque patient est placé dans la liste au bon endroit en fonction de la valeur de son attribut *Etat*.

```
Taille de la liste = 5
[0] : Patient : Amirault / Etat : 3
[1] : Patient : Cornier / Etat : 2
[2] : Patient : Nehaus / Etat : 2
[3] : Patient : Alves / Etat : 1
[4] : Patient : Drancé / Etat : 1

Taille de la liste = 4
[0] : Patient : Cornier / Etat : 2
[1] : Patient : Nehaus / Etat : 2
[2] : Patient : Alves / Etat : 1
[3] : Patient : Drancé / Etat : 1
(base) martindrance@martindrance-G5-5587:
```

Illustration 3: Affichage sur le terminal après compilation et éxécution du fichier main.cpp

Les surcharges des opérateurs < et = sont déclarées et définies dans la classe **Patient**, permettant à ces surcharges l'accès aux membres de la classe **Patient**. De plus, pour mener à bien cet exercice, il a fallut surcharger l'opérateur <<. En effet, dans la méthode *display()* de la classe **ListeOrd**, la méthode doit pouvoir afficher à l'écran des objets de la classe **Patient**. Il est dont nécessaire de re-définir le comportement de l'opérateur << pour qu'il puisse interagir avec des instances de **Patient**.