MOOC Intro POO C++
Exercices semaine 5

Exercice 16 : formes polymorphiques (niveau 1)

> Cet exercice correspond à l'exercice n°60 (pages 150 et 335) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Le but de cet exercice est de vous illustrer les problèmes qui peuvent se poser lorsque l'on veut manipuler des objets de façon polymorphique (un objet pouvant se substituer à un autre objet).

Il s'agit de manipuler quelques formes géométriques simples en leur associant une méthode qui affiche leur description.

16.1 Formes

Dans un fichier `formes.cc`, définissez une classe `Forme` en la dotant d'une méthode `void description()` qui affiche à l'écran : « `Ceci est une forme !` » .

Ajoutez au programme une classe `Cercle` héritant de la classe `Forme`, et possédant la méthode `void description()` qui affiche à l'écran : « `Ceci est un cercle.` ».

Recopiez ensuite la fonction `main()` suivante, et testez votre programme :

```
int main() {
 Forme f;
 Cercle c;
 f.description();
 c.description();
 return 0;
}
```

Ajoutez maintenant ceci à la fin de la fonction `main()` :

```
 Forme f2(c);
 f2.description();
```

Testez ensuite à nouveau votre programme.

Voyez-vous vu la nuance ?
Pourquoi a-t-on ce fonctionnement ?

Continuons nos prospections...

Ajoutez encore au programme une fonction :

```
void affichageDesc(Forme& f)
```

qui affiche la description de la forme passée en argument en utilisant sa méthode `description()`.

Finalement, modifiez le `main` ainsi :

```
int main() {
 Forme f;
 Cercle c;
 affichageDesc(f);
 affichageDesc(c);
 return 0;
}
```

Testez le programme... **Le résultat vous semble-t-il satisfaisant ?** Pourquoi?

Modifiez le programme (ajoutez 1 seul mot) pour que le résultat soit plus conforme à ce que l'on pourrait attendre.

16.2 Formes abstraites

Recopiez le programme précédent dans le fichier `formesabstraites.cc`, et modifiez la classe `Forme` de manière à en faire une **classe abstraite** en lui ajoutant la méthode virtuelle pure `double aire()` permettant de calculer l'aire d'une forme.

Écrivez également une classe `Triangle` et modifiez la classe `Cercle` existante héritant toutes deux de la classe `Forme`, et implémentant les méthodes `aire()` et `description()`.

Vous l'aurez deviné, la classe `Triangle` aura comme attributs `base` et `hauteur` ainsi qu'un constructeur adéquat, et `Cercle` aura
comme seul attribut `rayon` (ainsi qu'un constructeur adéquat).

Modifiez la fonction `affichageDesc` pour qu'elle affiche, en plus, l'aire de la forme passée en paramètre et testez avec la fonction main suivante :

```
int main() {
 Cercle c(5);
 Triangle t(10, 2);
 affichageDesc(t);
 affichageDesc(c);
 return 0;
}
```

Exercice 17 : encore plus de formes (niveau 2)

> Cet exercice correspond à l'exercice n°60 (pages 149 et 334) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Le but de cet exercice est d'arriver à définir une collection hétérogène de figures géométriques.

Nous nous limiterons, dans un premier temps, à une solution partielle... mais instructive.

Dans le fichier `figures.cc`, prototypez et définissez les classes suivantes : (vous pouvez adapter ce qui avait été réalisé dans l'exercice précédent).

- La classe abstraite `Figure`, possédant deux méthodes et aucun attribut :
  - `affiche()`, méthode publique, constante, virtuelle pure, et ne prenant aucun argument.
  - une méthode `Figure* copie() const`, également virtuelle pure, chargée de faire une copie en mémoire de l'objet et de retourner le pointeur sur cette copie
- Trois sous-classes (héritage publique) de `Figure` : `Cercle`, `Carre` et `Triangle`.
- Une classe nommée `Dessin`, qui modélise une collection de figures. Il s'agira d'une collection «hétérogène» d'éléments créés dynamiquement par une méthode ad-hoc définie plus bas (`ajouteFigure`).

Pour chacune des classes `Cercle`, `Carre` et `Triangle`, définissez les attributs (privés/protégés) requis pour modéliser les objets correspondants.

Définissez également, pour chacune de ces sous-classes, un constructeur pouvant être utilisé comme constructeur par défaut, un constructeur de copie et un destructeur.

Dans les trois cas, affichez un message indiquant le type de l'objet et la nature du constructeur/destructeur.

Définissez la méthode de copie en utilisant le constructeur de copie.

Finalement, définissez la méthode virtuelle `affiche`, affichant le type de l'instance et la valeur de ses attributs.

Ajoutez un destructeur explicite pour la classe `Dessin`, et définissez-le de sorte qu'il détruise les figures stockées dans la collection en libérant leur espace mémoire (puisque c'est cette classe `Dessin` qui, dans sa méthode `ajouteFigure`, alloue cette mémoire).

Comme pour les autres destructeurs, affichez en début de bloc un message, afin de permettre le suivi du déroulement des opérations.

Prototypez et définissez ensuite les méthodes suivantes à la classe `Dessin` :

- `void ajouteFigure(Figure const& fig)`

  qui ajoute à la collection une copie de la figure donnée en paramètre en faisant appel à sa méthode `copie`.

- `void affiche() const`

  qui affiche de tous les éléments de la collection.

Testez votre programme avec le main suivant:

```
int main() {
   Dessin dessin;

   dessin.ajouteFigure(Triangle(3,4));
   dessin.ajouteFigure(Carre(2));
   dessin.ajouteFigure(Triangle(6,1));
   dessin.ajouteFigure(Cercle(12));

   cout << endl << "Affichage du dessin : " << endl;
   dessin.affiche();
}
```

Compilez en ignorant pour l'instant les warnings. Testez le programme et assurez vous de bien comprendre l'origine de tous les messages affichés.

Si vous avez suivi les directives de l'énoncé, votre programme devrait, en dernier lieu, indiquer que le destructeur du dessin est invoqué... mais pas les destructeurs des figures stockées dans le dessin !
Pourquoi ?

Voyez-vous un moyen permettant de pallier cela ?

Corrigez (en ajoutant 1 seul mot) votre programme.

Ajoutez la fonction suivante, juste avant votre `main` :

```
void unCercleDePlus(Dessin const& img) {
   Dessin tmp(img);
   tmp.ajouteFigure(Cercle(1));
   cout << "Affichage de 'tmp': " << endl;
   tmp.affiche();
}
```

Appelez cette fonction depuis `main` (n'importe où après la première instruction d'ajout de figure).

Si tout se passe « normalement », le système doit interrompre prématurément votre programme, en vous adressant un message hargneux (genre «segmentation fault», ou «bus error»). (Il est cependant possible que sur certains systèmes cette erreur, qui se produit pourtant effectivement, ne soit pas détectée et ne donne lieu à aucun comportement visible.)

Quel est, à votre avis, le motif pour un tel comportement ? Corrigez votre programme.

Exercice 18 : Puissance 4 (niveau 3)

> Cet exercice correspond à l'exercice n°61 (pages 151 et 342) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

On envisage de programmer un jeu de puissance 4 (consistant à aligner quatre pions de même couleur soit horizontalement, soit verticalement, soit en diagonale). Le but de cet exercice est de fournir une première solution (sans interface graphique).

Pour programmer ce jeu, on distinguera trois types d'« objets » : des joueurs, un jeu, et une partie.

Un `Joueur` sera caractérisé par une `Couleur` (rouge ou jaune, par exemple), un nom et une méthode virtuelle pure `jouer(Jeu&)`.

Un `Jeu` sera une table carrée pouvant contenir des pions de `Couleur` ou être vide. On devra pouvoir donner la taille de la table lors de l'initialisation d'un `Jeu` (valeur par défaut : 8).

On fournira également une méthode `jouer()` prenant un numéro de colonne et une couleur. Lorsque cela est possible (colonne non pleine), cette méthode ajoute, dans la colonne correspondante et à la position libre la plus basse, le pion de la couleur passée en paramètre. Elle retourne un booléen indiquant si le pion a pu être placé ou non.

On ajoutera de plus à la classe `Jeu` une méthode `gagnant` qui retourne la couleur du gagnant, s'il y en a un, et « vide » sinon.

La classe `Partie` sera constituée d'un tableau de deux joueurs (dont le premier est celui qui commence) et d'un `Jeu`.

Pour pouvoir effectivement jouer, il faut maintenant implémenter des joueurs. Considérons deux types de joueurs :

- les joueurs humains, pour lesquels la méthode `jouer` consiste à afficher le `Jeu`, à demander à l'utilisateur le numéro de colonne à jouer, jusqu'à ce que le coup soit valide ; puis à le jouer effectivement sur le jeu ;
- l'ordinateur, pour lequel dans le cadre de cet exercice la méthode `jouer` sera très simple : jouer la première position possible rencontrée. L'étudiant intéressé pourra bien entendu compliquer à souhait cette méthode et implémenter des stratégies plus évoluées.

  On informera l'utilisateur du coup joué par un message à l'écran.

  Cette méthode dans sa version la plus simple ne contient donc vraiment que quatre lignes de code !

Tester le programme en créant une partie dans le `main`, avec un `Joueur` humain et un `Joueur` ordinateur (lequel commence).

Pour cela la classe `Partie` devra avoir une méthode `lancer` qui fait jouer les joueurs tour à tour, qui vérifie à chaque fois s'il y a un gagnant ou si le jeu est plein, et qui affiche le résultat en conséquence.

Exemple de déroulement

```
Entrez votre nom :
Corinne Titgoute

Le programme a joué en 1


#
--------
12345678
Joueur Corinne Titgoute, entrez un numéro de colonne
  (entre 1 et 8) : 12
-> Coup NON valide.
Joueur Corinne Titgoute, entrez un numéro de colonne
  (entre 1 et 8) : 1
Le programme a joué en 1




#
O
```

```
#
--------
12345678

Joueur Corinne Titgoute, entrez un numéro de colonne
  (entre 1 et 8) :
```

Exercice 19 : librairie (niveau 2)

> Cet exercice correspond à l'exercice n°72 (pages 183 et 381) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Un libraire souhaite gérer sa base de données d'ouvrages à l'aide d'un programme (écrit par vous en C++).

19.1 La classe Livre (POO de base)

Codez dans `librairie.cc` une classe `Livre` dotée des attributs suivants : `titre` (string), `auteur` (string), nombre de `pages` (int) ainsi qu'un attribut indiquant si l'ouvrage est un `bestseller` ou pas.

Dotez votre classe :

- d'une méthode `calculer_prix` permettant de calculer le prix d'un livre. Ce prix est calculé comme étant le nombre de pages multiplié par 0.3 auquel on ajoute 50 francs si c'est un bestseller.
- d'un constructeur prenant comme arguments des paramètres permettant d'initialiser tous les attributs de la classe.
- d'un destructeur virtuel vide
- d'une méthode `afficher` affichant la valeur des attributs d'un livre. Le prix du livre ne sera pas affiché par cette méthode. Inspirez-vous de l'exemple de l'exécution donné ci-dessous pour réaliser vos affichages.

Les attributs de votre classe seront privés. Les définition des méthodes de votre classe se feront à l'extérieur du corps de la classe.

Testez votre classe au moyen d'un petit `main` de sorte à produire quelque chose comme :

```
titre : Harry Potter à l'école des sorciers
auteur : J.K. Rowling
nombre de pages : 308
bestseller : oui
prix : 142.4

titre : Le fou
auteur : Gogol
nombre de pages : 252
bestseller : non
prix : 75.6
```

19.2 Des sous-classes de Livres (Héritage)

Continuez en modifiant les droits d'accès des attributs privés de la classe `Livre` de sorte à ce qu'ils soient accessibles directement dans les sous-classes de `Livre`.

Codez ensuite les sous-classes suivantes de la classe `Livre` :

1. `Roman` : caractérisé par un attribut indiquant s'il s'agit d'une `biographie` ou pas. Cet attribut sera protégé.
2. `BeauLivre` (ouvrages illustrés à la présentation soignées) : pas d'attributs spécifiques

La classe `Roman` possède elle aussi une sous-classe, `Policier`, modélisant les romans policiers. Cette sous-classe n'a pas d'attributs spécifiques. Vous la coderez également.

Les sous-classes ne doivent pas restreindre les droits d'accès des attributs et méthodes hérités des classes `Livre` et `Roman`.

La méthode `calculer_prix` codée dans la classe `Livre` donne le prix *standard* d'un livre.

Dotez vos sous-classes des méthodes publiques suivantes :

1. Classe `Roman` :
   - un constructeur prenant en arguments les paramètres permettant d'initialiser l'ensemble des attributs de la classe `Roman`.
   - un destructeur virtuel vide
   - une méthode `afficher` permettant d'afficher la valeur de l'ensemble des attributs d'un roman. Le prix du roman ne sera pas affiché par cette méthode. Inspirez-vous de l'exemple d'exécution ci-dessous pour réaliser vos affichages.
2. Classe `BeauLivre` :
   - un constructeur. Ce dernier devra prendre en arguments les paramètres permettant d'initialiser l'ensemble des attributs hérités de `Livre`.
   - un destructeur virtuel vide
   - un méthode `calculer_prix` calculant le prix d'un `BeauLivre` comme le prix standard augmenté de 30 francs. Par

exemple un `BeauLivre` de 100 pages (prix standard 30 francs) et qui est un best-seller (prix standard 30+50 = 80 francs) se vendra 110 francs (= 80 + 30 francs).

3. Classe `Policier` :
   - un constructeur. Ce dernier devra prendre en arguments les paramètres permettant d'initialiser l'ensemble des attributs hérités de la classe `Roman`.
   - une méthode `calculer_prix` calculant le prix d'un roman `Policier` comme le prix standard auquel on enlèvera de 10 francs. Si le prix ainsi obtenu est négatif, le prix de l'ouvrage sera de 1 franc.
   - un destructeur virtuel vide

Le calcul du prix dans les méthodes `calculer_prix` redéfinie dans les sous-classes `Policier` et `BeauLivre` doit faire appel à la méthode `calculer_prix` (prix standard) héritée de la classe `Livre`.

La méthode `afficher` de la classe `Roman` doit également faire appel à celle de la classe `Livre` afin d'éviter la duplication inutile du code commun.

Testez votre programme au moyen du `main` de sorte à produire quelque chose comme :

```
titre : Le chien des Baskerville
auteur : A.C.Doyle
nombre de pages : 221
bestseller : non
Ce roman n'est pas une biographie
prix: 56.3

titre : Le Parrain
auteur : A.Cuso
nombre de pages : 367
bestseller : oui
Ce roman n'est pas une biographie
prix: 150.1

titre : Le baron perché
auteur : I. Calvino
nombre de pages : 283
bestseller : non
Ce roman n'est pas une biographie
prix: 84.9

titre : Mémoires de Géronimo
auteur : S.M. Barrett
nombre de pages : 173
bestseller : non
Ce roman est une biographie
prix: 51.9

titre :  Fleuves d'Europe
auteur : C. Osborne
nombre de pages : 150
bestseller : non
prix: 75
```

19.3 Une collection de Livres (Polymorphisme)

19.3.1 Collection de livres

Codez maintenant une classe `Librairie` ayant un attribut privé nommé `livres`. Cet attribut est une collection de `Livres` codée comme un tableau dynamique.

Prototypez et définissez ensuite les méthodes suivantes de la classe `Librairie` :

- `void ajouter_livre(Livre*)` qui ajoute un (pointeur sur) un `Livre` la collection
- `void afficher() const` faisant appel à la méthode `afficher()` (de la classe `Livre` ou de ses classes dérivées) sur chaque élément de la librairie
- `void vider_stock()` permettant de supprimer tous les livres stockés dans la collection, tout en libérant leur espace mémoire.

19.3.2 Affichage polymorphique

Modifiez la fonction `afficher` associée à la classe `Livre` de sorte à ce qu'elle affiche aussi le prix du livre. Le calcul du prix à afficher se fera au moyen de la fonction `calculer_prix()` .

Dans la classe `Librairie`, la méthode `afficher` de la classe Livre est invoquée de façon polymorphique sur les éléments d'une collection de livres.

Modifiez les déclarations des méthodes `afficher` relatives aux livres de sortes à ce qu'elle puissent faire l'objet d'une résolution dynamique des liens. Faites-en de même avec les méthodes `calculer_prix` (car `calculer_prix` est invoquée par afficher).

En principe, votre fichier `librairie.cc` contient maintenant tous les outils nécessaires pour traiter les objets de la collection hétérogène de façon polymorphique.

Testez votre code au moyen du `main` de façon à obtenir :

```
titre : Le chien des Baskerville
auteur : A.C.Doyle
nombre de pages : 221
bestseller : non
prix: 56.3
Ce roman n'est pas une biographie

titre : Le Parrain
auteur : A.Cuso
nombre de pages : 367
bestseller : oui
prix: 150.1
Ce roman n'est pas une biographie

titre : Le baron perché
auteur : I. Calvino
nombre de pages : 283
bestseller : non
prix: 84.9
Ce roman n'est pas une biographie

titre : Mémoires de Géronimo
auteur : S.M. Barrett
nombre de pages : 173
bestseller : non
prix: 51.9
Ce roman est une biographie

titre : Fleuves d'Europe
auteur : C. Osborne
nombre de pages : 150
bestseller : non
prix: 75
```

à partir du code suivant :

```
int main()
{
  Librairie l;

  l.ajouter_livre(new Policier("Le chien des Baskerville", "A.C.Doyle",
                               221, false, false));
  l.ajouter_livre(new Policier("Le Parrain ", "A.Cuso", 367, true, false));
  l.ajouter_livre(new Roman("Le baron perché", "I. Calvino", 283, false, false));
  l.ajouter_livre(new Roman ("Mémoires de Géronimo", "S.M. Barrett", 173, false, true));
  l.ajouter_livre(new BeauLivre ("Fleuves d'Europe", "C. Osborne", 150, false));

  l.afficher();
  l.vider_stock();

  return 0;
}
```