TP Séance 9
Hiérarchie Formes géométriques

Le but de ce TP est de compléter puis de manipuler par polymorphisme une hiérarchie d'héritage de taille conséquente en C++. Nous nous appuierons sur un exemple de hiérarchie de classes de formes géométriques introduite en TD.

1 La hiérarchie de classes de formes

Une partie de cette hiérarchie est déjà implémentée : récupérez chez vous le répertoire (et la totalité de son contenu) `~/Bibliotheque/M2103-POO/TP09`.

Etude de la hiérarchie et du code C++ fourni

Placez vous dans le répertoire `version_1`.

Exercice 20 : Le but ici est de se familiariser avec l'ensemble du code fourni. Pour cela :

1. Consultez le fichier `A_LIRE.txt` qui vous servira de guide.
2. Consultez le code de l'ensemble des fichiers des classes des formes, et dessinez sur votre feuille la hiérarchie d'héritage mise en oeuvre.
3. Compilez, testez, lisez le reste du code, etc... En particulier, soyez sûrs d'avoir bien compris le rôle de la classe `Screen` et de la manière de l'utiliser dans les autres classes et dans le `main`.

Rajout de classes dans la hiérarchie

On souhaite compléter la hiérarchie avec 2 classes de formes : `Triangle` qui hérite de la classe abstraite `Shape`, et `Square` qui hérite de `Rectangle`.

Exercice 21 : Implémentez ces deux nouvelles classes :

1. complétez les fichiers vides à votre disposition, `Triangle.*` et `Square.*`,
2. et testez les en décommentant leur utilisation dans le `main`.
3. Les fichiers `saisies.*` contiennent des fonctions de saisie des formes qui seront utiles pour la suite. Rajoutez les fonctions de saisie pour les deux nouvelles formes que vous venez d'implémenter.

2 Gestion d'un dessin

Avant de commencer cette partie, recopiez dans le répertoire `version_2` les fichiers des deux classes `Triangle` et `Square`. vous trouverez dans ce répertoire (en plus des classes de la hiérarchie de formes) :

— `GraphicScene.*` qui sont vides,
— `main.cc` à décommenter et à compléter,

On souhaite donc écrire ici un programme permettant de créer et de manipuler un ensemble de formes dessinées à l'écran, en interaction avec un utilisateur via un menu.

Pour cela, il faut une structure de données qui permet de stocker les formes créées pour pouvoir à tout moment les afficher, les modifier, etc... Nous proposons d'utiliser un **ensemble de pointeurs sur `Shape`**. Cet ensemble sera mis à jour à chaque ajout/suppression de formes, et sera parcouru "par polymorphisme" pour l'affichage ou toute modification des formes.

La classe `GraphicScene`

Pour rendre le code plus lisible et plus modulaire, on propose de gérer cet ensemble de pointeurs sur `Shape` au sein d'une classe dont voici une entête (minimum) :

```
class GraphicScene
{
  private:
    set<Shape *> m_shapes;

  public:
    GraphicScene();
    ~GraphicScene();
    void addShape( Shape * pshape );
    void refresh( Screen & s );
};
```

La méthode `addShape` sert à rajouter un pointeur à l'ensemble de `Shape`, et la méthode `refresh` à afficher l'ensemble des formes à l'écran. Voici un exemple d'utilisation classique de ces méthodes si la variable `dessin` est de type `GraphicScene` :

```
dessin.addShape( new Line{black, Point(4,5), Point(15,22)} );
dessin.refresh( ecran );
```

Exercice 22 : Implémentez cette version minimum de la classe `GraphicScene`. Cela vous permettra ainsi d'obtenir les fonctionnalités 1 à 6 du menu proposées dans le `main`. Implémentez une à une ces fonctionnalités d'ajout des 5 formes et de leur affichage à l'écran.
Remarque : pensez que les fichiers `saisies.*` contiennent des fonctions de saisie des paramètres pour les formes, utilisez les !

3 Pour aller plus loin

Modifications s'appliquant à toutes les formes (c, d et e dans menu)

Exercice 23 : Rajoutez à la classe `GraphicScene` les 3 méthodes suivantes :

```
void setColourAll( char col );
void moveAll( int dx, int dy );
void eraseAll();
```

et utilisez les pour implémenter les options c, d et e du menu.

Modification s'appliquant seulement à une forme (x, y et z dans menu)

Ces fonctionnalités se réalisent en deux étapes :

1. sélection d'une forme,
2. application de la modification sur cette forme.

L'étape de sélection d'une forme peut être réalisée de plusieurs façons différentes, par exemple en fonction d'un identifiant unique associé à chaque forme, ou selon l'appartenance ou non d'un point à cette forme.

Exercice 24 : Modifier la hiérarhice de classes afin d'ajouter à une forme, un identifiant qui lui est propre.

Exercice 25 : Avec une sélection par l'identifiant.
Rajoutez à la classe `GraphicScene` la méthode `Shape * select( int id ) const` qui retourne le pointeur sur la forme d'identifiant `id`, ou `NULL` s'il n'existe pas. Il suffit alors d'appliquer la bonne fonction sur le pointeur de forme obtenu pour implémenter les options x, y et z du menu.
Remarque : il y a une subtilité pour l'option z, il faut aussi modifier l'ensemble de formes... donc, une nouvelle fonction dans la classe `GraphicScene` serait bien utile pour effacer "proprement" la forme d'identifiant `id` du dessin : `void erase( int id )` .