T.P.6: Gestionnaire de formes géométriques

Développer un programme capable d'instancier, de manipuler et de dupliquer des formes géométriques en deux dimensions.

La description des patrons de classes utilisés dans ce TP (`std::pair` , `std::vector` , `std::string` et `std::map`) est disponible sur le site de référence cplusplus.com

A l'issue de la séance de T.P.

- *Fournir un compte-rendu manuscrit répondant aux questions Q1 à Q10.*

A. Définition des formes 2D

Les fonctionnalités d'une forme géométrique sont les suivantes:

- La création d'une forme n'est possible que si l'on fournit l'ensemble des attributs qui la caractérisent.
- Une forme peut retourner son périmètre.
- Une forme peut afficher ses caractéristiques.

Dans le cadre de ce TP, on se limite à deux types de formes géométriques: les polygones fermés et les cercles.

Un polygone fermé est caractérisé par les coordonnées 2D de chacun des points qui le constituent. Un cercle est caractérisé par les coordonnées 2D de son centre ainsi que par son rayon.

L'implémentation en C++ est réalisée par le biais des classes `Polygone` et `Cercle`, toutes deux dérivées de la même classe de base `Forme` (cf. Figure 1).

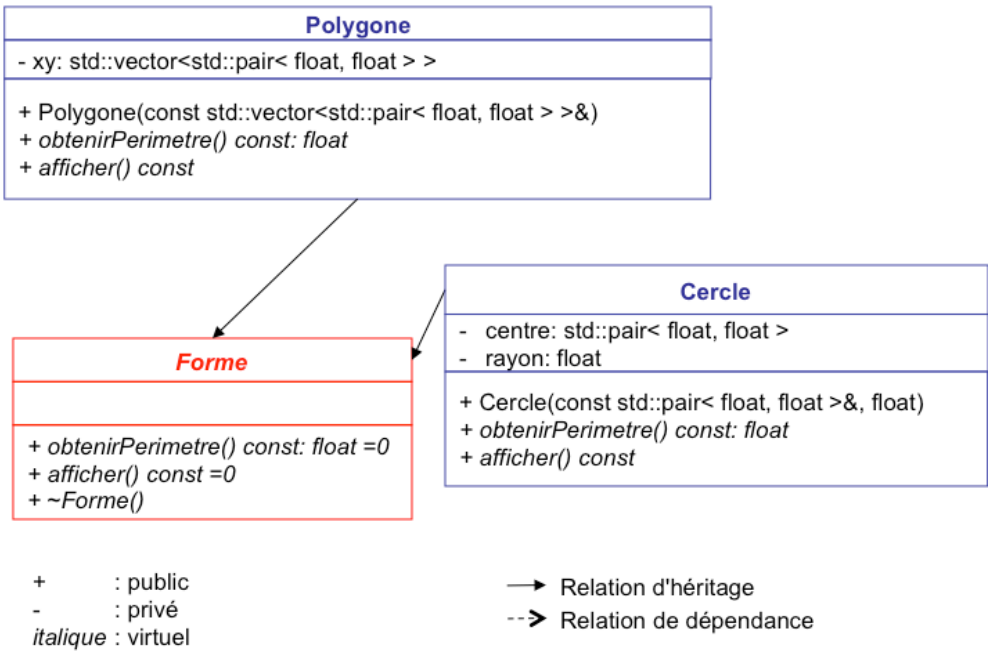

Figure 1: diagramme de classes, partie A.

Q1. Que permettent de représenter les classes `std::pair<float,float>` et `std::vector<std::pair<float,float> >` ?

Q2. Pourquoi n'a t'on pas défini de constructeur pour la classe `Forme` ?

Q3. Pourquoi n'est il pas possible d'instancier des objets de la classe `Forme` ?

Le code source de ces 3 classes étant fourni (cf. fichiers *Forme.h*, *Polygone.h* et *.cpp*, *Cercle .h* et *.cpp*), instancier et manipuler des cercles et des polygones fermés en mettant en œuvre **le polymorphisme**.

Exécuter le programme avec `valgrind` afin de rechercher d'éventuelles fuites mémoire.

On souhaite ajouter la fonctionnalité suivante aux formes géométriques :

- Une forme peut être translatée dans le plan 2D.

Q4. Par quel biais implémenter ce comportement sachant que la translation doit pouvoir être exprimée sous la forme de deux variables de type `float` ou d'un objet de type `std::pair<float, float>` ?

Compléter le code source existant et tester.

B. Fabrication en série

On souhaite à présent compléter les fonctionnalités du programme afin d'être capable à tout moment de:

- mémoriser une copie de certains des cercles et polygones instanciés,
- réaliser des copies des cercles et polygones mémorisés.

Nous allons pour cela définir une nouvelle classe `Usine` répondant au mode de fonctionnement suivant:

"*Une usine possède des objets et associe à chacun d'eux une clé unique permettant de l'identifier. On peut à tout moment ajouter un objet dans l'usine en lui attribuant une clé spécifique, et supprimer un objet de l'usine en indiquant sa clé. Il est possible d'obtenir de l'usine l'ensemble des clés des objets qu'elle possède. L'usine est capable de fournir la copie d'un objet dont on lui aura spécifié la clé.*"

Q5. Pourquoi mémoriser des copies des cercles et polygones plutôt que simplement leur adresse ?

L'implémentation en C++ est réalisée par le biais de la classe `Usine` qui exploite la hiérarchie de classes précédente afin de manipuler des objets `Cercle` et `Polygone` (cf. Figure 2).

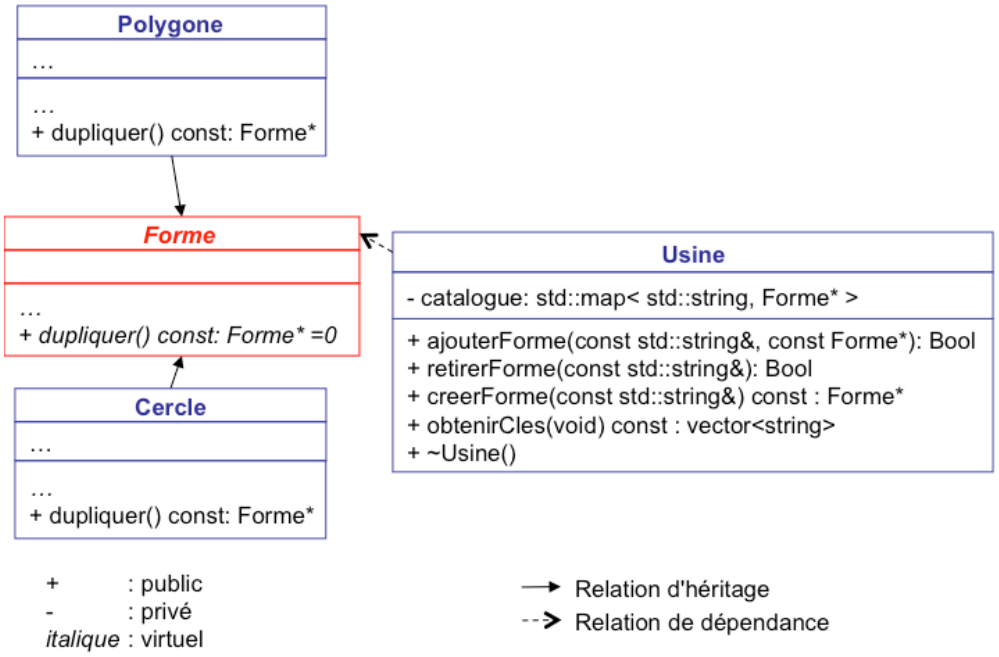

Figure 2: diagramme de classes, partie B.

Q6. Que permet de représenter la classe `std::map<std::string, Forme*>` ?

Q7. Pourquoi manipuler des cercles et des polygones par le biais de pointeurs de la classe `Forme` ?

Q8. Que doivent réaliser les méthodes `Cercle::dupliquer()` et `Polygone::dupliquer()` ? Dans quelle(s) méthode(s) devront elles être appelées ?

Q9. Comment réagir lorsque l'utilisateur souhaite créer un objet dont la clé n'est pas référencée ?

Q10. Quelles opérations faut-il réaliser lorsque l'usine est détruite ?

Ecrire le code source de la classe `Usine` et compléter celui des classes `Forme`, `Polygone` et `Cercle`.

Développer un programme de test et de validation le plus complet possible, sans erreurs d'exécution et respectant impérativement le pseudo-code suivant :

> *Instanciation d'une usine U, de cercles $C_{i=0...N}$ et de polygones $P_{j=0...M}$*
> *Ajout des cercles $C_{i=0...N}$ et polygones $P_{j=0...M}$ dans l'usine U*
> *Destruction des cercles $C_{i=0...N}$ et des polygones $P_{j=0...M}$*
>
> *Exploitation de l'usine U*