TP 3 - Formes géométriques

Objectifs :

— mise en œuvre héritage et gestion des constructeurs ;
— utilisation composant prédéfinis (classe **`string`**) ;
— utilisation assistant création de classes de l'IDE ;
— polymorphisme.

Introduction

Vous allez implémenter une série de classes permettant de faire de la géométrie basique. Outre la classe **`Point`**, vue précédemment, on va créer des classes permettant de représenter diverses formes géométriques et en particulier des rectangles et des triangles. Ces formes partagent un certain nombre de points communs, notamment :

— leur nom,
— le point "origine" de la forme,
— une surface,
— un périmètre,
— la capacité d'être affichée.

On va donc créer une classe racine des ces formes, qui prendra naturellement le nom **`Forme`**. On l'utilisera ensuite comme classe de base pour en faire dériver les classes **`Rectangle`** et **`Triangle`**.

1 Création des classes `Forme` et `Point`

En utilisant l'explorateur Windows, créer dans son dossier **`T:\Info3\TP3`**. Lancer votre IDE et créer un nouveau projet de type "Console", nommé **`tp3`**.

1.1 Classe `Point`

1. Reprendre depuis le dossier "TP2" la classe **`Point`** : copier les deux fichiers **`point.h`** et **`point.cpp`** et les coller dans le dossier TP3. Ensuite dans l'IDE, les ajouter au projet (fenêtre à gauche (*project*), clic-droit → Add to project). Remplacer ensuite les attributs **`int`** par des **`float`**, ainsi que dans les arguments du constructeur à deux arguments.

2. Ajouter à cette classe une méthode **`void Deplace( float, float )`** qui va permettre de déplacer un point. On pourra utiliser cette méthode de la façon suivante, par exemple :

```
Point pt(3,4);
cout << "avant: "; pt.Print();
pt.Deplace(2,4); // le point se retrouve en 5,8
cout << "après: "; pt.Print();
```

Ajouter le code ci-dessus dans une fonction **`main()`** et vérifier que le point passe bien en 5,8.

1.2 Classe `Forme`

1. Sélectionner l'onglet « Classes » dans la fenêtre de gauche et faire « clic-droit → Nouvelle Classe ». Un dialogue-assistant vous propose de paramétrer la création de cette classe. Indiquer le nom de la classe (**`Forme`**) et laisser les autres champs avec les valeurs par défaut. Après validation, l'IDE a ajouté deux fichiers à votre projet qui implémentent cette classe, en ne lui donnant pour l'instant que deux méthodes : un constructeur sans arguments et un destructeur, toutes les deux vides.

2. Ajouter à cette classe un attribut **`p0`** de type **`Point`** (point "origine" de cette forme) et un attribut **`nom`**, de type **`string`**. Cette classe générique de la bibliothèque standard C++ est déclarée dans le fichier d'en-tête **`string`** et est définie dans "l'espace de nom" (*namespace*) **`std`**. Il faudra donc avoir au début du fichier de déclaration de la classe la ligne :

```
#include <string>
```

et la déclaration se fera en précisant l'espace de nom :

```
    ...
    std::string nom;
    ...
```

Ces deux attributs n'ont pas à être accessibles par les classes dérivées, ils seront donc placés dans la section "**`private`**" de la déclaration de la classe.

3. Ajouter à cette classe un deuxième constructeur, qui aura comme argument un point et une chaîne :
`Forme( std::string, Point )`

4. Ajouter dans **`main()`** une ligne pour créer une forme en mémoire, en lui donnant un nom :

```
Point pt0(4,5);
Forme fo( "ma_forme", pt0 );
```

L'expression ci-dessus va appeler le constructeur à deux arguments de **Forme** et lui transmettre
- La valeur de **pt0**,
- La chaîne de caractère, convertie en objet de type **string**.

Dans le corps du constructeur, il faudra simplement assigner au point-origine de la forme la valeur du point transmis et assigner à l'objet "nom" la chaîne transmise.

5. Ajouter à la classe **Forme** une méthode **void Affiche()**, qui devra envoyer sur cout le nom de la forme, puis afficher le point via l'utilisation de la méthode **Print()** de la classe **Point** (voir TP2).

6. Vérification : valider le fonctionnement de ces classes avec la fonction **main()** suivante :

```
int main()
{
    Point pt0( 4, 5 );
    Forme fo( "ma_forme", pt0 );
    fo.Affiche();
}
```

2 Construction de classes héritées

1. En utilisant l'assistant vu ci-dessus, créer deux classes **Rectangle** et **Triangle**, mais **bien préciser** dans le dialogue qu'elles héritent de **Forme**. Indiquer dans l'assistant les arguments du constructeur, qui seront :
   - pour un triangle : un string (nom de la forme) et trois points
   - pour un rectangle : un string, un point (origine) et deux float (largeur et hauteur)

   Pour le rectangle, le dialogue doit être complété comme indiqué sur la fig. 1. Procéder ensuite de façon similaire pour le triangle.
   Vous penserez à ajouter dans les fichiers .cpp la ligne
   `using namespace std;`.

2. Ajouter (en "privé") à la classe **Triangle** deux attributs **pt1** et **pt2** de type **Point** (le troisième étant constitué par le "point-origine" de **Forme**) et à la classe **Rectangle** trois attributs de type **Point**, appelés **pt1**, **pt2**, **pt3**.

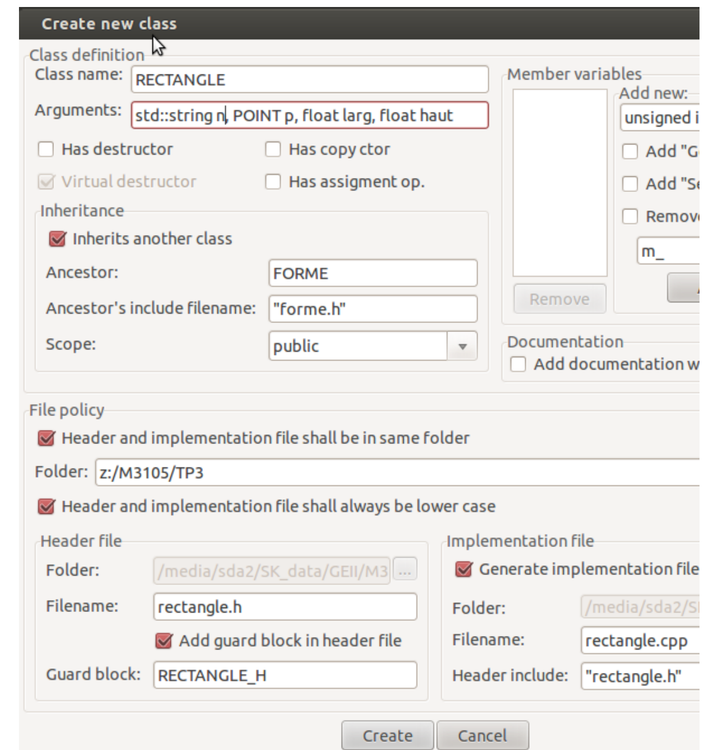

FIGURE 1 – Assistant de création de classe

3. Éditer les définitions des constructeurs de ces deux classes de façon à appeler **automatiquement** le constructeur **Forme(string, Point)** et lui "passer" les arguments transmis, pour initialiser le nom de la forme et le point à l'origine. Dans le corps du constructeur, il faudra aussi initialiser les attributs propres à la forme, soit, pour le **Triangle** :
   `pt1 = p1; pt2 = p2;`
   Pour le rectangle, comme on transmet largeur et hauteur, il faudra d'abord initialiser les trois autres points à la valeur du point à l'origine :
   `pt1 = pt2 = pt3 = p;`
   Puis déplacer chacun de ces points, via l'appel de la méthode **Deplace()**, en utilisant les arguments **larg** et **haut**.

   Validation : on pourra valider la création des objets avec le programme suivant :

```
int main()
{
      Triangle t1( "t1", Point(1,1), Point(3,3), Point
          (4,5) );
      Rectangle r1( "r1", Point(1,3), 7, 2 );
      t1.Affiche();
      r1.Affiche();
}
```

Redéfinition de méthodes dans les classes dérivées En l'état, le fonctionnement n'est pas satisfaisant : l'appel de `Affiche()` exécute la méthode de la classe de base, nous privant de l'affichage de la totalité de l'information contenue dans l'objet (les coordonnées des autres points).

4. Ajouter a chacune des deux classes `Rectangle` et `Triangle` une méthode `Affiche()`, qui devra d'abord afficher "Rectangle" ou "Triangle", puis faire un appel à la méthode `Affiche()` de la classe de base (affichage du nom et du point à l'origine). Il faudra ensuite afficher les coordonnées des autres points qui composent la forme.

5. Recompiler le programme et vérifier le fonctionnement.

3 Mémorisation d'objets hétérogènes dans un tableau

On est parfois amené à manipuler dans un même ensemble des objets de nature différentes. En C++, le conteneur `vector` de la bibliothèque standard se prête bien à ce genre de chose. Cependant, on ne peut stocker que des des objets de nature identique. La solution consiste à ranger dans le tableau non pas les objets, mais l'adresse de stockage en mémoire des objets, c'est à dire mémoriser une liste de **pointeurs**.

1. Créer dans la fonction `main()` un `vector` de pointeurs de type `Forme`, ayant 4 éléments : `vector<Forme*> tab(4);`
(Pensez à ajouter `#include <vector>` dans le fichier `main.cpp`) :

2. Ajouter la création de deux rectangles et de deux triangles (nommés `r1`, `r2`, `t1`, `t2`) et les ranger dans le tableau. Ne pas oublier l'opérateur "adresse de" (&) :

```
tab[0] = &t1;
tab[1] = &t2;
tab[2] = &r1;
tab[3] = &r2;
```

3. Ajouter ensuite une boucle `for(...)` et afficher les 4 formes stockées, via la méthode `Affiche()`.

4. Compiler et exécuter. Quel est le problème rencontré ? Quelle méthode `Affiche()` est exécutée ?

5. Quelle est la solution ? (voir cours !). Modifier la classe `Forme`, recompiler tout le projet (CTRL-F11) et re-tester le programme.

4 Périmètre et surface

1. Ajouter à la classe `Forme` :
   — deux attributs réels `Perimetre` et `Surface` (en "protected")
   — la déclaration d'une méthode virtuelle `void Affiche_PS()` qui devra calculer et afficher leur valeur.

   La définition de cette méthode (dans `forme.cpp`) se contentera d'afficher "Périmètre et Surface non disponible". En effet, on ne peut pas généraliser leur calcul pour toutes les formes. Si ce calcul n'est pas implémenté par la classe de l'objet, cet affichage l'indiquera.

2. Ajouter un appel de cette méthode dans la méthode `Affiche()` de `Forme` et vérifier que le message apparaît bien en exécutant le programme précédent.

3. Ajouter à la classe `Rectangle` la déclaration et la définition de la méthode `Affiche_PS()`. Cette méthode devra (1) faire le calcul et (2) afficher les valeurs. Le calcul utilisera bien sûr la fonction `Distance()` pour obtenir largeur et hauteur à partir des points.
Il faudra également basculer l'attribut `p0` de la classe `Forme` en `protected`, de façon à pouvoir y accéder dans les méthodes `Affiche_PS()`.

4. Vérifier le fonctionnement : on doit avoir l'affichage du périmètre et de la surface pour le rectangle, mais **pas pour le triangle**.

5. Ajouter à la classe `Triangle` la méthode `Affiche_PS()`. La surface pourra se calculer avec la formule de Heron :

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

Avec p le demi-périmètre : $p = \frac{1}{2}(a+b+c)$