TD 2 – Premières classes

Préparation : Exercice 1

Exercice 1 : Préparation, modélisation UML

Voici le diagramme de la classe **`Point`** ainsi que sa définition (.h) et son implémentation (.cxx). Cette classe permet de représenter et de manipuler un point du plan. Le but sera de manipuler des formes géométriques basées sur cette classe.

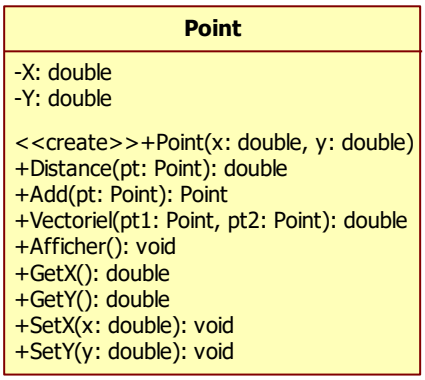


Diagramme de la classe Point

```
#ifndef _Point_h_
#define _Point_h_



class Point
{
private:
    double X;
    double Y;

 public:

 Point(double x = 0, double y = 0);

 Point Add(const Point &pt) const;

 void Afficher() const;

 double GetX() const;
 double GetY() const;

 void SetX(double x);
 void SetY(double y);

};


#endif
```

TD_Point.h

```
#include <iostream>
#include <math.h>
#include "TD_Point.h"

using namespace std ;

Point::Point(double x, double y)
{
 X=x;
 Y=y;
}

Point Point::Add(const Point &pt) const
{
 Point result(pt.X + X, pt.Y + Y);
 return result;
}

void Point::Afficher() const
{
 cout<< "(" << X << " ," << Y << " )";
}

double Point::GetX() const
{ return X; }

double Point::GetY() const
{  return Y; }

void Point::SetX(double x)
{  X = x; }

void Point::SetY(double y)
{  Y = y; }
```

TD_Point.cxx

1-a Analyser le programme suivant : comprendre chaque ligne du code et donner les affichages produits.

```
#include <iostream>
#include "TD_Point.h"

using namespace std ;

int main( void )
{
cout << " #### exo 1 #### " << endl;

Point pt1(1,1);
Point pt2;
Point * _pt;

pt2 = Point(3,1);
_pt = new Point();

cout << "Position de pt1 : ";
pt1.Afficher();
pt1.SetX( 2 ); pt1.SetY( 3 );
pt1.Afficher();
cout << endl;

cout << "Position de pt2 : ";
pt2.Afficher();
cout << endl;

*_pt = pt1.Add(pt2);

cout << "Position de pt3 : ";
_pt->Afficher();
cout << endl;

delete _pt;

cout << "Appuyer sur Entree pour continuer" << endl;
cin.get();
return 0;
}
```

Programme à analyser

1-b Donner le code C++ de la fonction membre « **`Distance`** » de la classe **`Point`** qui calcule la distance entre deux points : c'est-à-dire entre le point courant et un point passé en paramètre à la fonction.

1-c Modéliser (UML) une classe **`Triangle`**, permettant de définir un triangle dans le plan à partir de 3 objets **`Point`**. Donner à cette classe les fonctionnalités suivantes :

- Les fonctions d'accès en lecture et écriture aux champs (les 3 points).
- *Affichage* : pour afficher les coordonnées des 3 points.
- *Perimetre* : qui retourne le périmètre du triangle.
- *Translation* : qui translate le triangle par un vecteur (utiliser un objet Point pour le vecteur).
- *Aire* : qui retourne l'aire du triangle (formule de Héron ou ½ périmètre).

$$Aire = \sqrt{p.(p-a).(p-b).(p-c)} \quad avec\ p = (a+b+c)/2$$

Avec a, b et c les longueurs des cotés

1-d Modéliser en UML (ne pas implémenter les fonctions) la classe **`Quadri`** reprenant toutes les fonctionnalités de la classe **triangle**, mais pour un quadrilatère… Un quadrilatère sera défini par 4 points (les cas « papillon » ne seront pas considérés).

1-e Modéliser en UML (ne pas implémenter les fonctions) une classe **`Cercle`** reprenant toutes les fonctionnalités de la classe triangle, mais pour un cercle… Un cercle sera défini par un point « centre » et un rayon.

1-f Identifier dans les classes `Triangle`, `Carre` et `Cercle` les points communs : champs et fonctions membres. Comment modifier ces classes pour rendre les fonctions membres semblables (même algorithme) ? Existe-t-il une (ou plusieurs) fonction(s) membre des trois classes non compatible avec votre schéma ?

1-g A partir des réponses précédentes, proposer une modélisation hiérarchisée de cette application.

Exercice 2 : Réalisation, premières classes

2-a Valider les résultats de la question **1-a** et créer un projet incluant les fichiers `TD_main.cxx`, `TD_Point.h` et `TD_Point.cxx`. Ces 3 fichiers sont sur le réseau (*moodle → GE → support de cours → Informatique → Moo*) **dans le sous-dossier "Sources_Question_2a"** de l'archive TD_2.zip

NB : Pour valider le déroulement du programme (constructeurs, pointeurs, appels des fonctions membres, …), utiliser le débogueur et au besoin modifier ces fichiers.

2-b Implémenter la fonction membre « **`Distance`** » de la classe **`Point`** (question **1-b**) et tester cette méthode dans le programme.

2-c Implémenter votre classe **`Triangle`** modélisée à la question **1-c**.

… et héritage

2-d Voici le diagramme de classes proposé pour mutualiser le code (correspondant à une solution des questions **1-d**, à **1-g**).

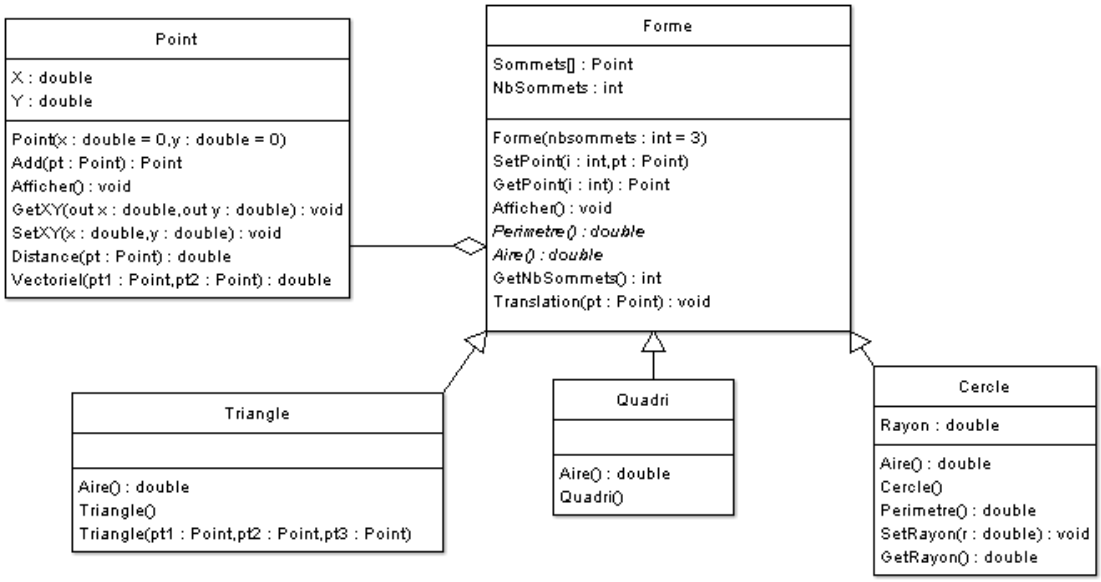


Diagramme de l'application

Ecrire en C++ les nouvelles classes **`Quadri`** et **`Cercle`** en vous appuyant sur les classes **`Point`**, **`Forme`** et **`Triangle`** fournies sous Moodle (`TD_Point.h`, `TD_Point.cpp`

`TD_Triangle.h`, `TD_B_Triangle.cpp`, `TD_Forme.h`, `TD_Forme.cpp`), ainsi qu'un exemple d'utilisation (`TD_main.cpp`). Vous trouverez ces fichiers **dans le sous-dossier "Sources_Question_2d"** de l'archive TD_2.zip.

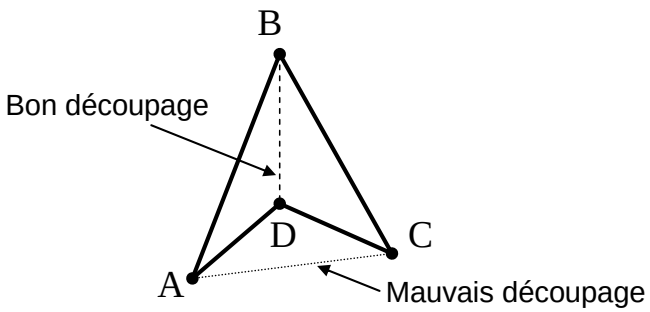
Remarques pour le calcul de l'aire des quadrilatères : (on ne considère pas les quadrilatères papillons, ni triangles...)

- Un quadrilatère peut se décomposer en 2 triangles et ce, de 2 manières différentes.
- Dans le cas des quadrilatères **concaves**, un seul des 2 découpages conduit au bon calcul de l'aire. Pour trouver le bon découpage il faut trouver les 2 points non connexes dont les 2 angles pour passer d'un point à l'autre sont de signes opposés. Exemple, dans la figure ci-dessous les 2 découpages possibles, et les triangles obtenus, sont :
  - segment de coupe [AC], triangles (C, A, D) et (C, A, B) : mauvais découpage,
  - segment de coupe [BD], triangles (B, D, A) et (B, D, C) : bon découpage.

Les angles pour le découpage suivant [AC] sont ADC et ABC : ils sont de même signe
➔ mauvais découpage.

Les angles pour le découpage suivant [BD] sont DAB et DCB : ils sont de signes opposés
➔ bon découpage.

Exemple de quadrilatère concave et des découpages pour le calcul de l'aire :



… Ainsi on pourra montrer que :

- pour un quadrilatère non-concave, la somme des aires obtenues pour chaque découpage est la même ;
- pour un quadrilatère concave, la somme des aires du bon découpage est plus petite que celle obtenue par le mauvais découpage…