18,5/20 : TB

```
/***********************
Cercle.h
***********************/
#pragma once
#define _USE_MATH_DEFINES
#include <math.h>
#include "Point2D.h"
#include "Figure.h"

//#define PI 3.14159265358979323846

class Point2D;
class Figure;

class Cercle : public Figure
```

1 point

```
{
private :
        double rayon;
        Point2D centre;

public:
        Cercle(double leRayon, Point2D leCentre);
        double getPerimetre();
        double getSurface();
};



/***********************
Commande.h
***********************/
#pragma once
#include <vector>
#include <string>
#include "Figure.h"

using namespace std;

class Figure;

class Commande
{
private:
        bool commandeTerminee;
        double prixMetreDecoupe , prixMetreCarreMatiere ;
        string idCommande;
        vector<Figure*> lesFigures;

public:
        Commande(string identifiantCommande , double lePrixMetreDecoupe , double lePrixMetreCarreMatiere);
        string getIdCommande() { return idCommande; }
        void ajouterNouvelleFigure(Figure *laFigure);
        void cloturerCommande();

        double getPrix() ;

};



/***********************
Figure.h
***********************/
#pragma once

class Figure
{
public:
```

0,5 point

```
        virtual double getPerimetre() = 0;
        virtual double getSurface() = 0;
};

/***********************
Point2D.h
***********************/
// Cette classe n'est pas à modifier
#pragma once

class Point2D
{
private:
        double x , y ;

public:
        Point2D(double x=0 , double y=0);
        double getX();
        double getY();
        void setX(double newX);
        void setY(double newY);
};



/***********************
Polygone.h
***********************/
#pragma once

#include <vector>
#include "Figure.h"
#include "Point2D.h"

using namespace std;

class Point2D;
class Figure;

#define abs(x) ( (x) >=0 ? (x) : -(x) )

class Polygone : public Figure
{
protected:
        vector<Point2D *> lesSommets;
        bool estFerme;
```

1 point

```
public:
        Polygone(void);
        static double distance( Point2D &p1,  Point2D &p2);
        void insereUnNouveauSommet(Point2D *leSommet, int position = -1);
        void fermeLePolygone();
        double getPerimetre();
        double getSurface();

};



/***********************
Cercle.cpp
***********************/
#include "Cercle.h"


Cercle::Cercle(double leRayon, Point2D leCentre)
{
        this->centre = leCentre;
        this->rayon = leRayon;
}
```

1 point

```
double Cercle::getPerimetre()
{
        return 2 * M_PI * rayon;
}

double Cercle::getSurface()
{
        return M_PI * rayon * rayon;
}


/***********************
Commande.cpp
***********************/
#include "Commande.h"


Commande::Commande(string identifiantCommande , double lePrixMetreDecoupe , double lePrixMetreCarreMatiere)
{
        this->idCommande = identifiantCommande;
        this->prixMetreDecoupe = lePrixMetreDecoupe;
        this->prixMetreCarreMatiere = lePrixMetreCarreMatiere;
}
```

this->commandeTerminee=false;

```
void Commande::ajouterNouvelleFigure(Figure* laFigure)
{
        lesFigures.push_back(laFigure);
}
```

2,5 points

```
void Commande::cloturerCommande()
{
        this->commandeTerminee = true;
}
```

Retourner 0 si la commande n'est pas terminée

```
double Commande::getPrix()
{
        double prixPer = 0, prixSurf = 0;
        for (unsigned i = 0; i < lesFigures.size(); i++) {
                prixPer = lesFigures[i]->getPerimetre() * prixMetreDecoupe;
                prixSurf = lesFigures[i]->getSurface() * prixMetreCarreMatiere;
        }
        return prixPer + prixSurf;
}



/**********************
main.cpp
**********************/
#include <iostream>
#include <conio.h>
#include "Polygone.h"
#include "Cercle.h"
#include "Commande.h"

using namespace std ;                          // espace de nommage standard



int main()
{
        // Testez la classe Cercle
        cout << "Cercle test : " << endl;
        Cercle C_test(4, { 0,0 });
        cout << "Le perimetre : " << C_test.getPerimetre() << " et la surface : " << C_test.getSurface() << endl << endl;




        // Testez la classe Polygone avec la figure de test du sujet
        cout << "Polygone test : " << endl;
        double Coordonnees[6][2]={ { 1 , 1 } , { 3 , 5 } , { 5 , 7 } , { 5 , 1 } , { 3 , 3 } , { 3 , 1 } };
        Polygone P_test;
        for (unsigned i = 0; i < 6; i++) {
                P_test.insereUnNouveauSommet(new Point2D(Coordonnees[i][0], Coordonnees[i][1]));
        }
        P_test.fermeLePolygone();
        cout << "Le perimetre : " << P_test.getPerimetre() << " et la surface : " << P_test.getSurface() << endl << endl;
```

pg principal de test du polygone de la figure : 2 points

pg principal de test du sapin de Noël : 1 point

```
        // Sapin de Noel et boules
        cout << "Sapin de Noel : " << endl;
        double CoordonneesSapin[15][2]={ { 2 , 2 } , { 5 , 4 } , { 3 , 4} , { 5 , 6 } , { 4 , 6 } , { 6 , 8 },
{ 8 , 6 } , { 7 , 6 },
                                                                       { 9 , 4 } , { 7 , 4} , { 10 , 2 } , { 6
.5 , 2 } , { 6.5 , 1 },  { 5.5 , 1 } , { 5.5 , 2 }};
        Figure *Sapin;
        Sapin = new Polygone;

        // Création du polygone sapin
        for (unsigned i = 0; i < 15; i++) {
                ((Polygone*)Sapin)->insereUnNouveauSommet(new Point2D(CoordonneesSapin[i][0], CoordonneesSapin[i
][1]));
        }
        ((Polygone*)Sapin)->fermeLePolygone();

        cout <<"superficie du sapin = " << Sapin->getSurface() << "  ";
        cout <<"Perimetre du sapin = " << Sapin->getPerimetre() << endl;



        cout << "Boules de Noel : " << endl;
        double CoordonneesCentreCercles[6][2] = { { 2.5 , 3.5 } , { 3.5  , 5.5 } , { 4.5 , 7.5 } , { 7.5  , 7.5
} , { 8.5 , 5.5 } , { 9.5  , 3.5 } };
        int i;

        // Création des 6 cercles
        for (i = 0; i < 6; i++) {
                Figure* Cto_string(i);
                Cercle Cto_string(i)(0.5, new Point2D(CoordonneesCentreCercles[i][0], CoordonneesCentreCercles[i
][1]);
                cout << "superficie du cercle " << i << " = " << Cto_string(i)->getSurface() << "  ";
                cout << "Perimetre du cercle " << i << " = " << Cto_string(i)->getPerimetre() << endl;
        }
```

Partie à revoir : ne déclarez pas les objets à l'intérieur de la boucle

```
        //...

        //// Création de la commande du Père Noel
        //...

        //// Ajout des figures (le sapin et les 6 cercles) à la commande
        //...

        //// Affichage du prix de cette commande
        //cout <<"\nCout de la commande : " << ... <<"  = " << ... <<" euros" << endl;



        _getch();       // on attend l’appui sur une touche
        return 0 ;      // fin du programme
}



/***********************
Point2D.cpp
***********************/
// Cette classe n’est pas à modifier
#include "Point2D.h"


Point2D::Point2D(double x , double y)
{
        this−>x = x;
        this−>y = y;
}

double Point2D:: getX()
{ return x ;}

double Point2D::getY()
{
        return y;
}

void Point2D::setX(double newX)
{
        x = newX;
}

void Point2D::setY(double newY)
{
        y = newY;
}


/***********************
Polygone.cpp
***********************/
#include <math.h>
#include "Polygone.h"


Polygone::Polygone(void)
{
        this−>estFerme = false;
}

double Polygone::distance(Point2D &p1, Point2D &p2)
{
        double dist = 0;
        dist = sqrt((p1.getX() − p2.getX()) * (p1.getX() − p2.getX()) + (p1.getY() − p2.getY()) * (p1.getY() − p
2.getY()));
        return dist;
}

void Polygone::insereUnNouveauSommet(Point2D* leSommet, int position)
{
        if (position == −1) {
                lesSommets.push_back(leSommet);
        }
        else {
                lesSommets.insert(lesSommets.begin() + position, leSommet);
        }
}

void Polygone::fermeLePolygone()
{
        this−>estFerme = true;
        lesSommets.push_back(lesSommets[0]);
```

```
}

double Polygone::getPerimetre()
{
        double per = 0;
        if (estFerme) {
                for (unsigned i = 0; i < lesSommets.size() - 1; i++) {
                        per += distance(*lesSommets[i], *lesSommets[i + 1]);
                }
        }
        return per;
}

double Polygone::getSurface()
{
        double surf = 0;
        for (unsigned i = 0; i < lesSommets.size()-1; i++) {
                surf += ((lesSommets[i]->getX()* lesSommets[i + 1]->getY())- (lesSommets[i + 1]->getX() * lesSom
mets[i]->getY()));
        }
        return 0.5 * abs(surf);
}
```

2 points