TP Semaine 2
Mise au point d'une classe Polygone

Nous nous proposons de définir une classe `Polygone` où le nombre de côtés n'est pas défini à l'avance (mais est cependant inférieur à un maximum donné). Pour représenter un polygone, nous allons utiliser un tableau de taille variable de `Point`. Vous trouverez dans `/net/exemples/AP2/TP02/` les fichiers `Point.cc` et `Point.h`.

Exercice 6 : Fichier en-tête `Polygone.h`

Ecrivez le fichier `Polygone.h`. Pour cette première question, vous intégrerez les déclarations des méthodes suivantes : saisie d'un `Polygone`, affichage un `Polygone` (méthode `toString()`), translation d'un `Polygone` (méthode `deplace(...)`) et ajout d'un sommet (instance de la classe `Point`) à un polygone. Pensez à déterminer les méthodes constantes ou non ainsi que les passages par référence constante.
Le diagramme de classes ci-dessous donne la liste de toutes les méthodes que vous aurez mises en oeuvre à la fin de ce TP.

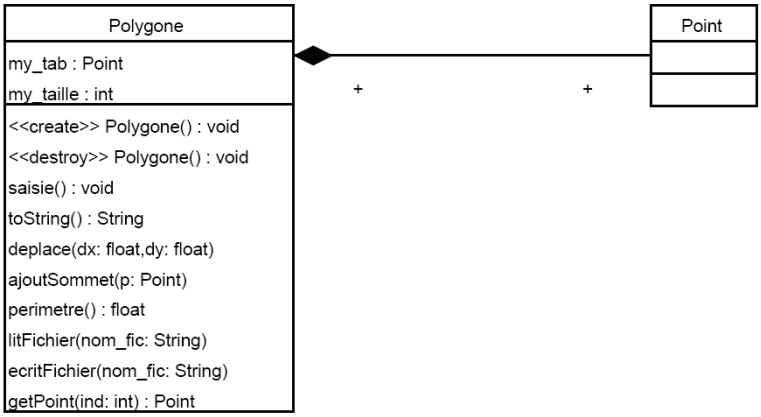

FIG. 1 – Interface de la classe Polygone.

Exercice 7 : Les autres fichiers

Ecrivez un fichier `Polygone.cc` dans lequel vous donnerez pour chacune des fonctions proposées dans l'interface une définition provisoire minimale, par exemple

```
void Polygone::saisie() {
    cout << "saisie..." << endl;
}
```

Cette démarche vous permet d'écrire « dans la foulée » un fichier `main.cc` contenant la fonction `main`, ainsi qu'un fichier `Makefile`. Vous pouvez ainsi vérifier ce que vous faites à mesure.

Exercice 8 : Méthodes de la classe `Polygone`

Définissez complètement les fonctions membres de la classe `Polygone`. Vous ne devez pas modifier la classe `Point`. Vous ferez attention à l'accès aux données membres de cette classe `Point`.

Exercice 9 : Périmètre d'un `Polygone`

Ajoutez à la classe `Polygone` une méthode permettant au `Polygone` de calculer son périmètre.

Exercice 10 : Lecture et écriture d'un `Polygone` dans un fichier

On considère un fichier dans lequel sont stockés des `Point`. On souhaite pouvoir affecter un `Polygone` à partir d'un tel fichier de `Point`, et aussi sauvegarder un `Polygone` dans un fichier.

1. Ecrivez une méthode pour la classe `Point` qui permet de lire un `Point` dans un flux passé en paramètre.
2. Ecrivez une méthode pour la classe `Polygone` qui prend en paramètre le nom d'un fichier de `Point`, et affecte au `Polygone` les `Point` qui sont lus dans ce fichier.
3. Ecrivez une méthode pour la classe `Polygone` qui permet de sauvegarder un `Polygone` dans un fichier. Quelle fonctionnalité faut-il ajouter à la classe `Point` ? Modifiez la classe `Point` en conséquence.

Exercice 11 : Récupérer un point d'un `Polygone`

Ajoutez à la classe `Polygone` une méthode `getPoint` permettant de récupérer le `Point` d'indice `ind` dans le `Polygone`. Les deux prototypes suivants sont possibles. Testez les deux, expliquez pourquoi et comment ça fonctionne.

```
Point getPoint( int ind ) const;
// ou
const Point & getPoint( int ind ) const;
```

Exercice 12 : Pour compléter proprement la classe `Polygone`...

Il faudrait rajouter au minimum :

- le constructeur de copie

- l'opérateur d'affectation

Surcharger aussi l'opérateur de flux `operator<<` serait également bien utile !

Pourquoi pas aussi d'autres opérateurs comme `operator==`, etc... A vous alors d'en définir la signification sur un `Polygone` et de l'implémenter dans chacune des méthodes.