

TP3 CPP

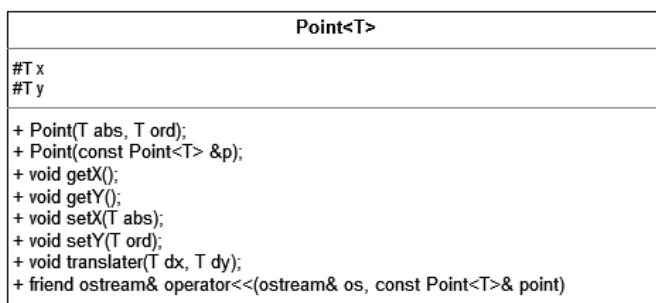
But :

L'objectif de ce TP est de manipuler des classes, de travailler sur la surcharge d'opérateurs et de mettre en œuvre le polymorphisme et utilisant des templates.

I. Réalisations des programmes

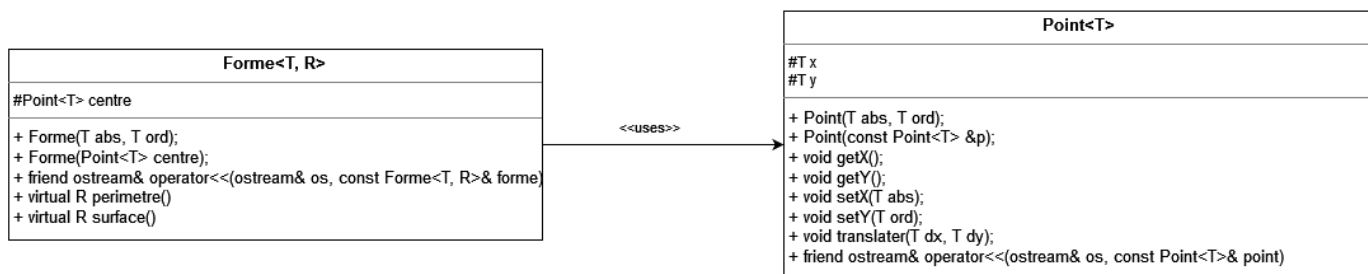
1. Création d'une classe Point template:

On définit une classe Point template défini par le diagramme de classe ci-dessous :



2. Création d'une forme géométrique abstraite

Nous réalisons une classe abstraite Forme représentant une forme géométrique et définissant deux méthodes abstraites perimetre et surface qui seront définies dans les classes concrètes qui héritent de Forme.



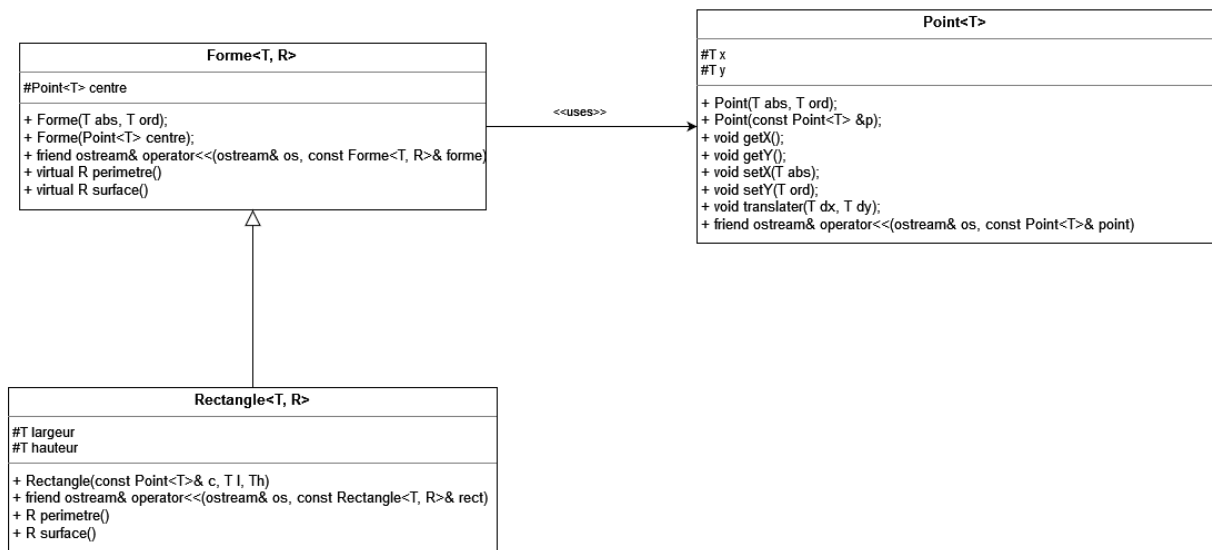
3. Création de formes géométriques concrètes

a. Classe Rectangle

Nous définissons ensuite une classe Rectangle qui hérite de Forme et qui comporte deux attributs supplémentaires hauteur et largeur.

Cette classe redéfinit les méthodes abstraites de Forme de sorte à retourner le périmètre et la surface de l'instance rectangle associé.

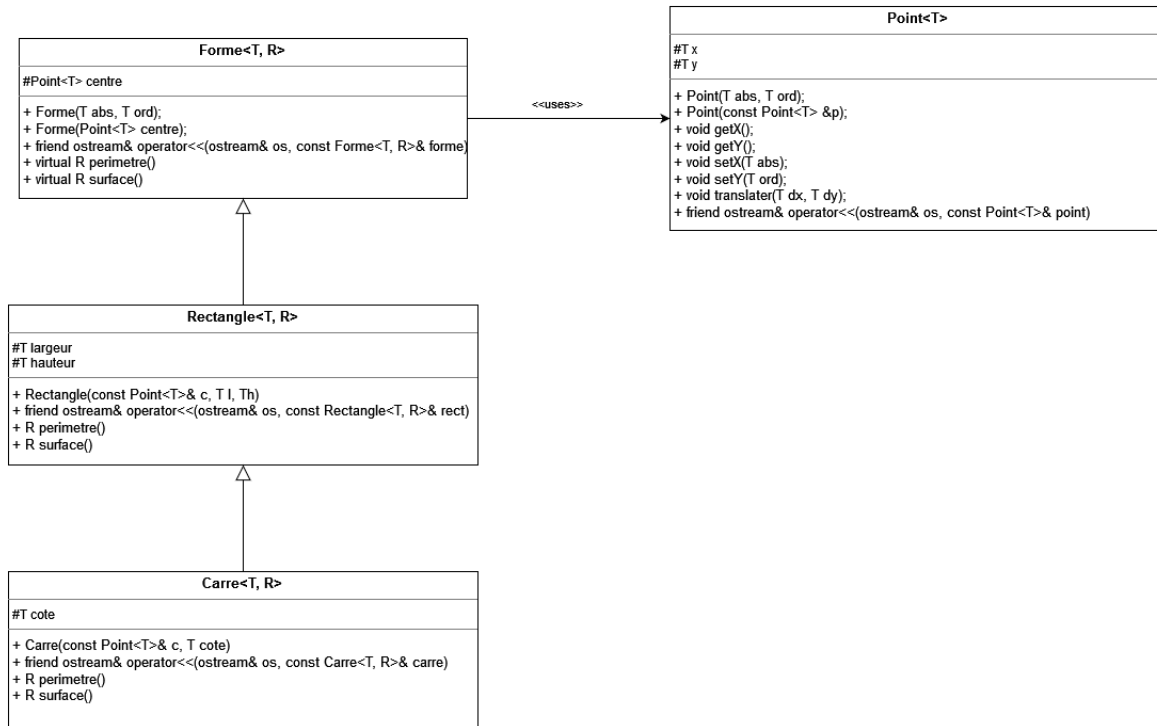
Le diagramme de classe de Rectangle est le suivant :



b. Classe Carré

Nous réalisons une classe Carré qui hérite de Rectangle et qui redéfinit les méthodes perimetre et surface.

Le diagramme de classe associé est le suivant :



4. Spécialisation

On spécialise la fonction `translater` de la classe `Point` pour les `string` de sorte à pouvoir translater des chaînes caractères. Pour ce faire, on utilise la fonction `swap`.

5. Liste de Formes

Nous définissons une classe `ListeFormes` qui permet de stocker des formes dans un tableau et permet de calculer la surface totale de la liste de formes et la boîte englobante de la liste de formes.

La classe `ListesFormes` implémentent les méthodes « ajouter » pour ajouter une forme à la liste, « `getFormes` » pour récupérer la liste de formes, « `surfaceTotale` » pour calculer la surface de toutes les formes et « `boiteEnglobante` » pour calculer le plus petit rectangle, aligné sur les axes, qui contient l'ensemble des formes de la liste.

II. Test : Jeux d'essais :

I. Point :

Nous créons une instance de la classe point ayant pour coordonnées (2.5 ; 3.5).

Nous affichons ce point puis le translatons de (1 ; -1).

```
TP 3 - Test de la classe Point
-----
Point p1: (2.5, 3.5)
Après translation de (1.0, -1.0), p1: (3.5, 2.5)
Point p2 (copie de p1): (3.5, 2.5)
-----
```

Le résultat de l'affichage confirme le bon fonctionnement de la classe Point. L'affichage est correct et la translation du point est bien réalisé. Lorsque l'on translate le point en (2.5 ; 3.5) de (1 ; -1), les nouvelles coordonnées du point sont (3.5, 2.5). Ce qui est bien cohérent. Au passage, on peut observer le bon fonctionnement du constructeur par recopie.

II. Forme

a. Rectangle

```
TP 3 - Test de la classe Rectangle
-----
Rectangle(Centre: (5.2, 4.3), Largeur: 10, Hauteur: 5)
Périmètre du rectangle: 30
Surface du rectangle: 50
-----
```

Nous créons ensuite une instance de la classe Rectangle ayant pour largeur 10 et hauteur 5 et observons le bon fonctionnement des méthodes perimetre et surface. Le résultat de l'affichage du perimetre et de la surface est cohérent. Le périmètre d'un rectangle de largeur 10 et de hauteur 5 est bien 30 tandis que sa surface est bien égale à 50.

b. Carre

```
TP 3 - Test de la classe Carre
-----
Carre(Centre: (5.2, 4.3), côté: 4)
Périmètre du carré: 16
-----
```

Nous créons ensuite une instance de la classe Carre ayant pour coté 4 et observons le bon fonctionnement des méthodes perimetre et surface comme pour l'instance de la classe Rectangle. Le périmètre d'un carré de coté 4 est bien égale à 16 comme sa surface. Cela confirme le bon fonctionnement de la classe et de ses méthodes.

III. Spécialisation

Nous créons un point avec pour coordonnées (« Hello », « World ») et appelons la méthode translater.

```
-----
TP 3 - Test de la spécialisation de Point pour std::string
-----
(Hello, World)
(World, Hello)
```

Les nouvelles coordonnées après la translation sont (« World », « Hello »). Ce qui montre bien que les chaînes de caractères sont échangées et que par conséquent la méthode translater agit différemment et de manière spécifique pour des données de types string.

IV. Liste de Formes

On crée une instance ListeFormes et on ajoute des formes créées précédemment :

- Un carré de 3cm de coté, de surface 9, de centre (5.2 ; 4.3)
- Un rectangle de longueur 4 et de largeur 6, de surface 24, de centre (7.9 ; 1.7)
- Un rectangle de longueur 2 et de largeur 8, de surface 16, de centre (-5.0 ; -5.0)

La surface totale de la liste est alors de 49.

```
TP 3 - Test de la classe ListeFormes
-----
Surface totale des formes: 49
Boîte englobante: Rectangle(Centre: (1.95, -1.6), Largeur: 15.9, Hauteur: 14.8)
PS C:\Users\Lucas\Desktop\Centrale TP\P00\tp3-lucas-djamael>
```

On constate que nous obtenons les bonnes valeurs pour la boîte englobante et la surface totale.