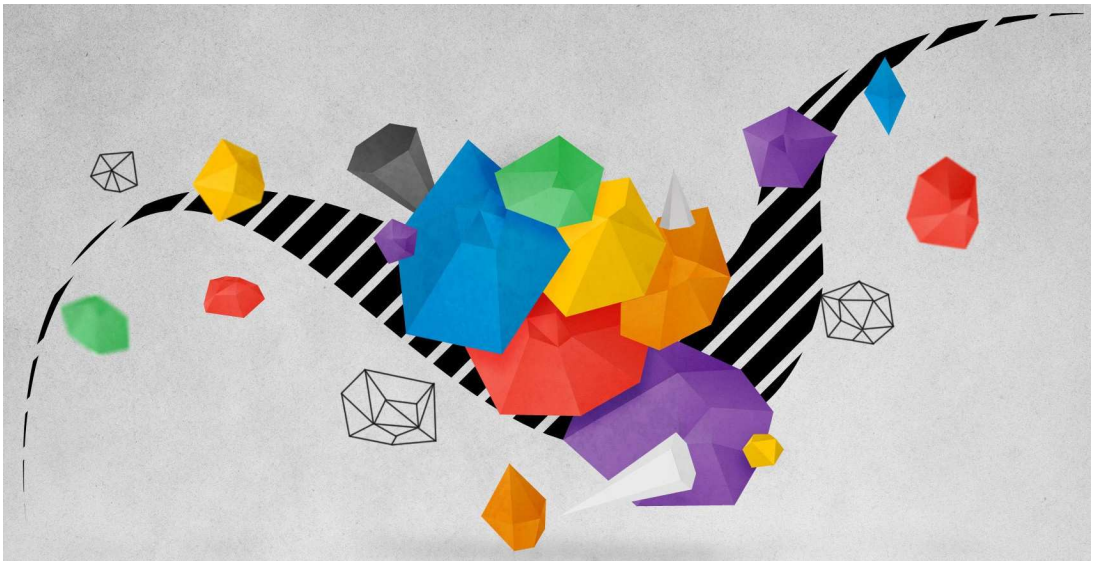
Niveau 3

TP
Développement logiciel
S4.6. - Programmation orientée objet
Modèle canonique de Coplien
Surcharges d'opérateurs

Formes, le retour

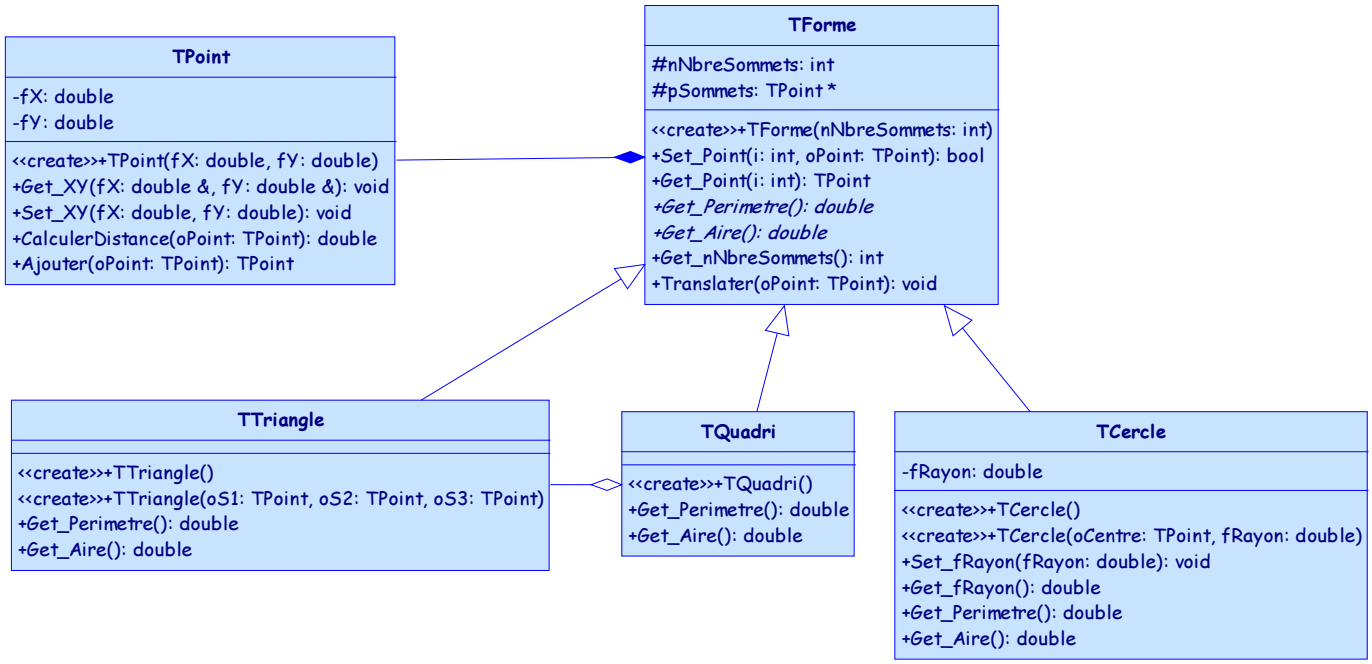
Table des matières



1 - PRÉSENTATION

Cette seconde partie du TP va nous permettre d'approfondir notre maîtrise de la notion d'héritage.

Le diagramme de classes étudié est toujours celui-ci :



Notre étude portera sur la partie héritage des classes `TTriangle`, `TQuadri` et `TCercle` à partir de la classe `TForme`.

2 - RETOUR SUR TFORME

Vous avez sûrement remarqué que deux des méthodes de `TForme` étaient indiquées en italique.

Il s'agit de méthodes **virtuelles**. Ce sont des méthodes de la classe mère qui peuvent être polymorphées dans les classes filles, c'est-à-dire qu'une méthode de la classe fille peut avoir le même prototype, faire un traitement équivalent, mais avoir un codage différent. Mais cela, n'est pas obligatoire.

Dans notre cas, les méthodes `Get_Perimètre` et `Get_Aire` sont **virtuelles**, car chaque classe fille va avoir sa propre façon de calculer le périmètre ou l'aire. Dans le fichier d'entête, le prototype serait :

```
virtual double  Get_Perimetre() ;
```

Mais est-ce que `TForme` sait comment calculer son périmètre à partir de son tableau de sommets ?

La réponse paraît être positive... mais en creusant un peu, une forme n'ayant qu'un sommet (comme un cercle) ou deux (comme...?) pose problème !

On ne peut donc pas coder, dans la classe `TForme`, la méthode `Get_Perimetre` !

On peut laisser son corps vide et penser que le développeur de la classe fille la codera correctement. Sauf que, si elle ne le fait pas, la méthode (vide) de la mère sera appelée et le résultat attendu sera, de fait, erroné !

On peut imposer au développeur de la classe fille de coder cette méthode sous peine de ne pas pouvoir compiler du tout !

Pour cela, on définit, dans la classe mère, que la méthode est **virtuelle pure** ! C'est-à-dire sans aucun code dans la classe mère (pas de définition du tout dans le fichier d'application).

Il suffit dans le fichier d'entête de prototyper la méthode virtuelle, en lui ajoutant `= 0` à la fin :

```
virtual double  Get_Perimetre()  = 0 ;
```

Notez qu'une méthode virtuelle pure est virtuelle avant tout (mot clef `virtual` obligatoire) !

Le problème avec une classe ayant au moins une méthode virtuelle pure, est qu'elle ne peut être instanciée (servir pour créer des objets) car toutes ses méthodes ne sont pas déclarées et codées !

Si une classe fille hérite d'une classe mère ayant au moins une méthode virtuelle pure, elle doit **obligatoirement** coder cette méthode pour pouvoir être instanciée ! Si la classe mère possède plusieurs méthodes virtuelles pures, toutes doivent être codées dans la fille !

Une classe ayant toutes ses méthodes virtuelles pures, est dite **classe virtuelle pure.**

Si elle n'en a que quelques-unes, mais pas toutes, elle est juste nommée **classe virtuelle.**

1) Modifiez la classe `TForme` afin d'implémenter le fait que les méthodes `Get_Perimetre` et `Get_Aire` soit virtuelles pures.

3 - TTRIANGLE

Contrairement à la classe `TPoint` qui a pu être reprise d'un précédent TP, la classe `TTriangle` doit-être recodée, car elle va faire intervenir l'héritage de la classe `TForme`.

2) Ajoutez au projet courant, la classe C++ `TTriangle`, en créant ses fichiers entête et application.

3) Dans le fichier entête, codez la déclaration de la classe, ainsi que l'héritage de `TForme`.

4) Ajoutez les méthodes nécessaires pour adapter la classe à la forme canonique de Coplien !

5) Tant qu'à y être, redéfinissez l'opérateur d'affichage `<<` de la classe `ostream`, pour afficher facilement les objets de cette classe !

6) Dans le fichier application, ajoutez les définitions des méthodes de `TTriangle`.

7) Complétez la définition de tous les constructeurs en stipulant qu'un triangle est une forme à 3 sommets obligatoirement.

8) Qu'y a-t-il à ajouter dans le destructeur ?

9) Déclarez des objets `oTriangle`, `oAutreTriangle`... dans le `main`, et testez la validité de tous les constructeurs et du destructeur.

10) Codez le corps de l'opérateur d'affichage qui sort à l'écran (même si le périmètre et l'aire ne sont pas encore connues !) :

```
Le triangle (adresse) a 3 sommets :
                Le point (adresse) est aux coordonnees [ X1 ; Y1 ]
                Le point (adresse) est aux coordonnees [ X2 ; Y2 ]
                Le point (adresse) est aux coordonnees [ X3 ; Y3 ]
                Son perimetre est P
                Son aire est A
```

11) L'opérateur d'affectation est nécessaire ! Codez-le !

12) Calculez le périmètre du triangle, et proposez la réponse en retour de la méthode `Get_Perimetre`.

13) Pour ce qui est de l'aire du triangle, on sait forcement la calculer de façon classique :

$$A = \frac{b \times h}{2}$$

mais dans notre cas, il n'est pas facile de déterminer la hauteur. Nous ne connaissons que les coordonnées des sommets, ou la longueur de chaque côté ! Tiens, d'ailleurs, la **formule de Héron** peut nous aider ! Elle permet de trouver l'aire d'un triangle à partir de la longueur de ses trois côtés `a`, `b` et `c` ainsi que de son demi périmètre `p` (ça, on sait faire, le périmètre) !

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

Codez la méthode `Get_Aire` !

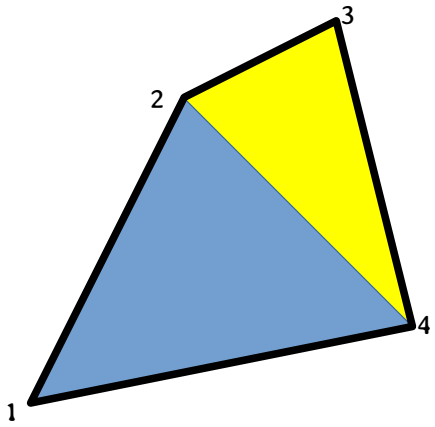
14) Vous avez, forcément, testé toutes ces méthodes dans le `main()`, grâce au débogueur !

4 - TQUADRI

On ne parlera pas, ici, de quadrilatère croisé. Le calcul de leur aire est plus complexe. Mais si vous y tenez, ce sera l'objet du bonus !

15) Sur le même modèle, codez la classe `TQuadri` qui est une forme ayant obligatoirement 4 sommets !

16) Pour ce qui est de l'aire d'un quadrilatère, il est possible de la calculer facilement à partir de celle de deux triangles :



Un premier délimité par les sommets 1, 2 et 4 du quadrilatère.

Le second par les sommets 2, 3 et 4.

La somme de ces deux aires donne l'aire totale du quadrilatère.

Codez la méthode `Get_Aire` en fonction des indications !

5 - TCERCLE

Un cercle est une forme n'ayant qu'un seul sommet : son centre. De plus, il est défini par un rayon (ou un diamètre... bon, le rapport est juste d'un demi entre ces deux grandeurs !)

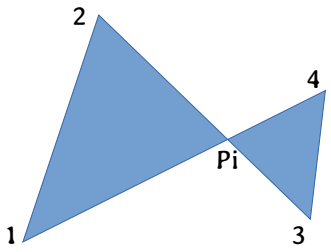
17) Sur le même modèle, codez la classe `TCercle`.

18) La méthode `Get_Perimetre` a besoin d'être adaptée ! Comment se calcule le périmètre d'un cercle ?

19) Codez la méthode `Get_Aire`. Facile !

6 - BONUS

20) Dans la classe `TQuadri`, arrivez à détecter si le quadrilatère est croisé ou non. Ajoutez une méthode de développement `EstCroise` qui retournera vrai ou faux en conséquence.



21) Ajoutez une méthode de développement `Get_Intersection` qui retourne le `TPoint`, nommé `Pi` sur le schéma, d'intersection des deux côtés croisés du quadrilatère.

22) Modifiez la méthode `Get_Aire` pour prendre en compte le cas d'un quadrilatère croisé.