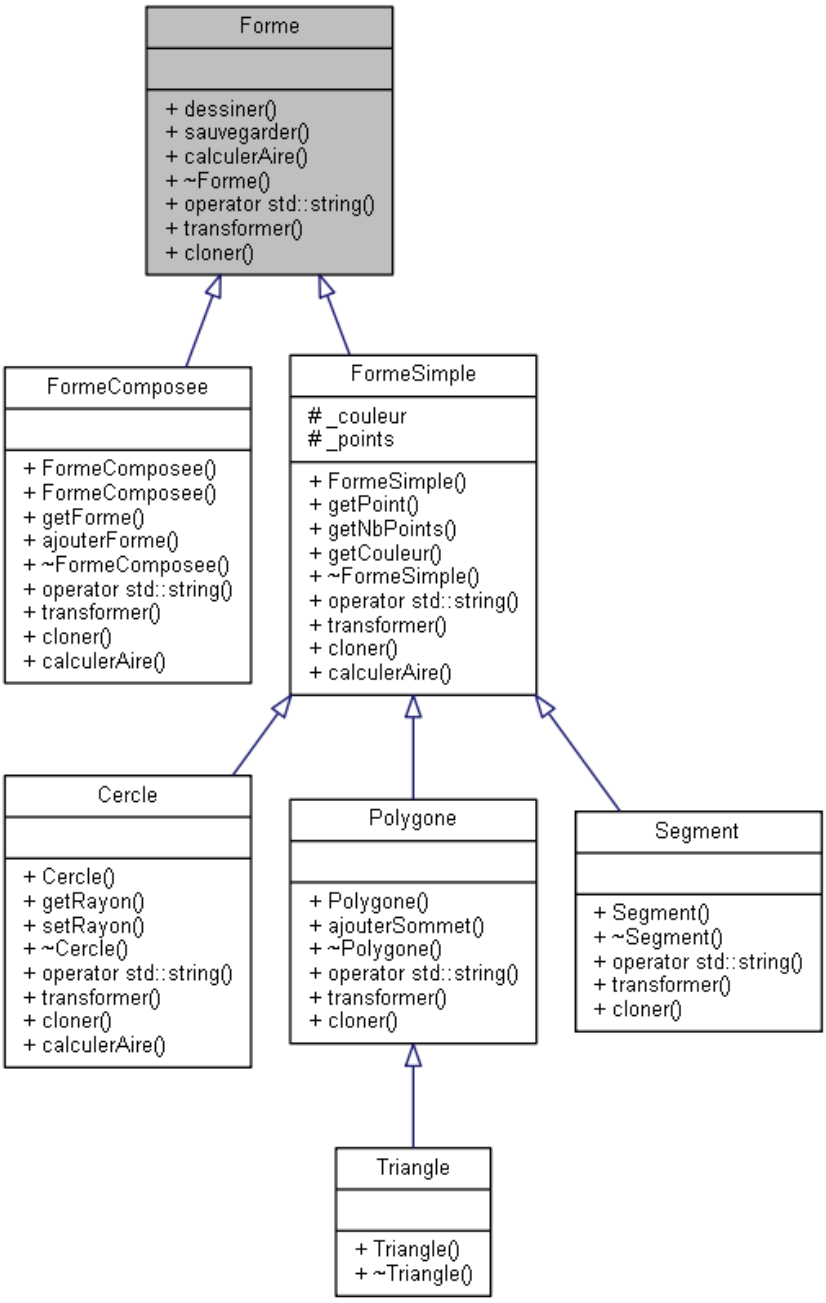
Client

Présentation

Le client a été développé en C++ comme imposé par le sujet. Il permet la création, manipulation, affichage et sauvegarde / lecture de plusieurs formes géométriques 2D.

Pour garder une cohérence avec tout le projet, toutes les couleurs, méthodes ou objets sont nommés en français.

Formes



La classe *Forme* est une classe abstraite représentant une forme géométrique. Elle définie les méthodes métiers disponibles sur toutes les formes à savoir :

- L'opération de dessin
- L'opération de sauvegarde
- Le calcul de l'aire
- La destruction
- La conversion en chaîne de texte
- L'opération de transformation
- L'opération de clonage

La classe *FormeComposee* représente un ensemble de formes. Elle dispose de méthodes propres comme :

- L'ajout de forme
- La récupération d'une forme la composant

La classe *FormeSimple* représente une forme géométrique 2D caractérisée par une couleur et un ensemble de points. Elle est virtuelle mais permet surtout de factoriser le code. Elle dispose de méthodes propres comme :

- La récupération d'un point la composant
- La récupération du nombre de points la composant
- La récupération de la couleur

La classe est également responsable du calcul de l'aire de toutes les autres formes en héritant (excepté pour la classe Cercle qui redéfinie le calcul d'aire). La formule utilisée pour le calcul de l'aire est la suivante :

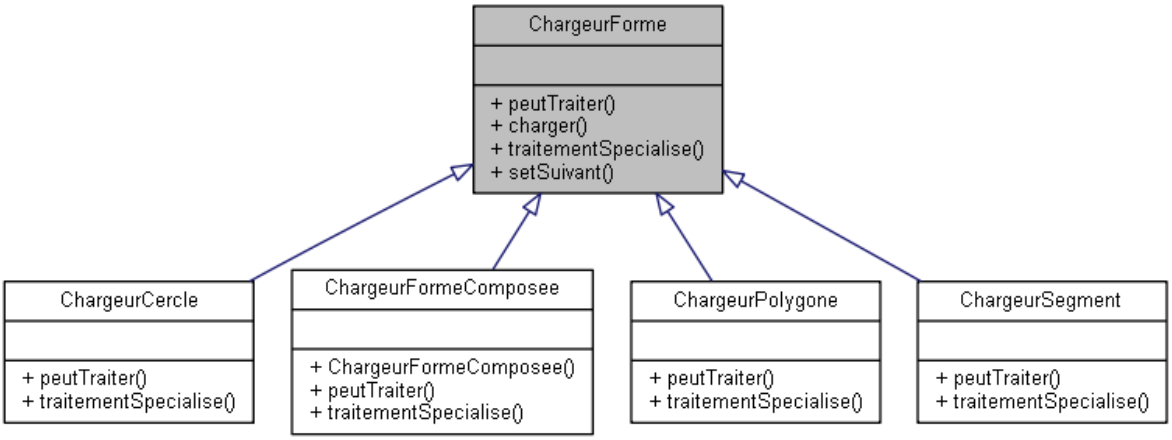
$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

Les classes *Cercle*, *Segment* et *Polygone* implémentent toutes la classe *FormeSimple*. Chaque classe redéfinie les méthodes au besoin comme :

- L'opération de transformation
- Le clonage
- La conversion en chaîne de texte

La classe *Triangle* hérite de *Polygone* et permet d'en créer un facilement. Elle ne redéfinie aucune méthode puisque les traitements sont identiques à ceux d'un polygone. L'affichage d'un triangle provoque l'affichage d'un polygone, aucune distinction n'est faite pour la sauvegarde ou l'envoi vers le serveur graphique par exemple.

Chargeur de formes

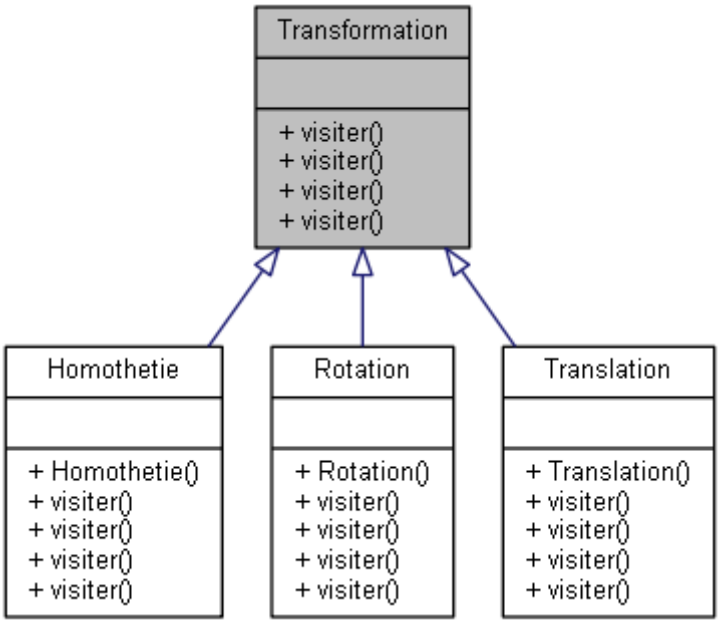


La classe *Forme* est une classe abstraite représentant un chargeur. La classe est en réalité une implémentation du design pattern chaîne de responsabilité.

Les méthodes *peutTraiter()* et *traitementSpecialise()* sont définies dans les classes filles. Les autres méthodes sont uniquement définies dans la classe mère.

Chaque classe est responsable du chargement de sa forme et renvoie une *Forme*.* La classe *ChargeurFormeComposee* prend un paramètre à l'instanciation : la chaîne de responsabilité. Ainsi, pour charger une forme composée, on se sert de la chaîne actuelle que l'on appelle sur chaque morceau de forme.

Transformations

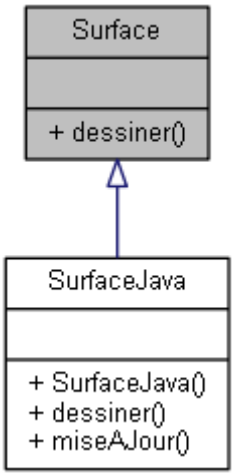


La classe *Transformation* est une classe abstraite représentant une transformation quelconque sur une forme. La classe est en réalité une implémentation du design pattern chaîne de responsabilité. Ici seules quelques transformations sont implémentées : homothetie, rotation et translation. On

pourrait envisager d'implémenter d'autres transformations comme une recoloration, une distorsion, …

Il y a quatre méthodes *visitier()* qui prennent chacune une forme différente : segment, cercle, polygone ou triangle.

Surface



La classe *Surface* représente une … surface. Dans le client seule la classe *SurfaceJava* a été implémentée. Il s'agit d'une connexion avec le serveur Java.

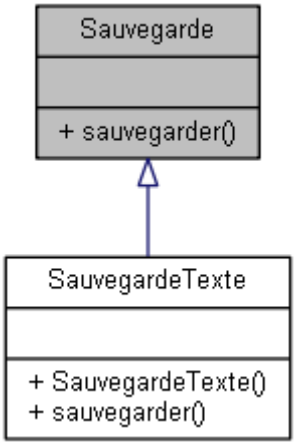
La méthode *dessiner()* agit comme un buffer : chaque forme est ajouté dans la chaîne de texte à envoyer au serveur.

Lorsque le client estime qu'il est temps d'afficher les formes a l'écran, il appelle *miseAJour()* qui se charge d'envoyer le message grâce à une connexion réseau.

Les sockets sont gérées grâce à la classe *Socket* qui est multiplateforme (testée sous Windows 10 et Ubuntu 18.04).

Cette approche a été envisagée car il est facile d'ajouter plusieurs surfaces comme SFML, libgdx, Qt, …

Sauvegarde



La classe *Sauvegarde* est une classe abstraite permettant de sauvegarder les formes sous différents formats. Ici, seule la classe SauvegardeTexte est implémentée.

Il faut fournir un chemin vers un fichier et la sauvegarde texte enregistre la représentation textuelle de la forme dans le fichier.

Il est aisé d'ajouter d'autres formats de sauvegarde tel que XML, JSON, binaire, …

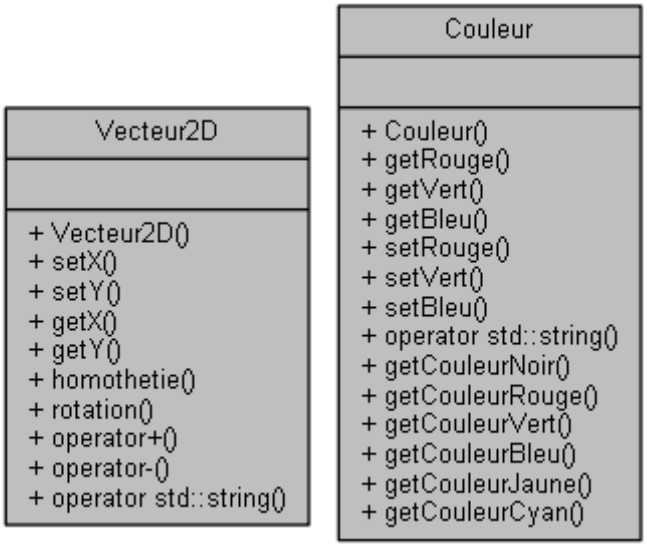
Socket

La classe *Socket* représente une connexion réseau. La classe implémente le design pattern singleton. Le constructeur se charger d'initialiser WinSock sur Windows et le destructeur décharge WinSock.

La compatibilité multiplateforme est assuré avec des directives de compilation.

À la création d'une socket, celle-ci se connecter à la cible.

Classes annexes



Ces deux classes facilitent l'écriture du code. Il n'y a pas de spécificité dans ces classes, elles réalisent un travail de base.