Rapport de Projet de Synthèse

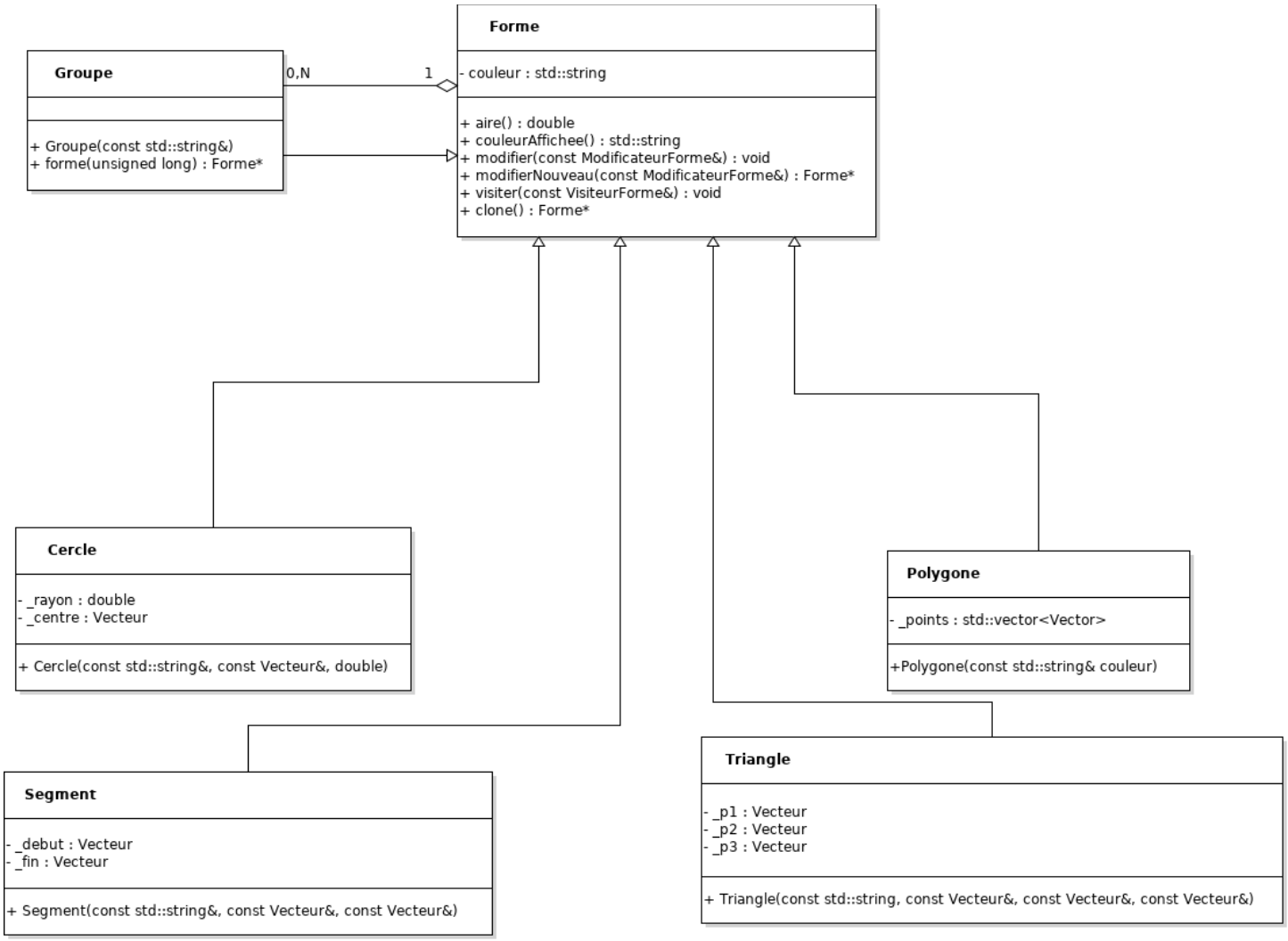
par Roméo Florian et Webert Chris

Ce projet permet d'afficher des formes géométriques sur un serveur distant.

Le client à été développé en C++11 sur Clion et Visual Studio, sur Windows et Linux.

Le serveur à été développé en Java 8 sur IntelliJ et Eclipse, sur Windows et Linux.

Représentation des formes



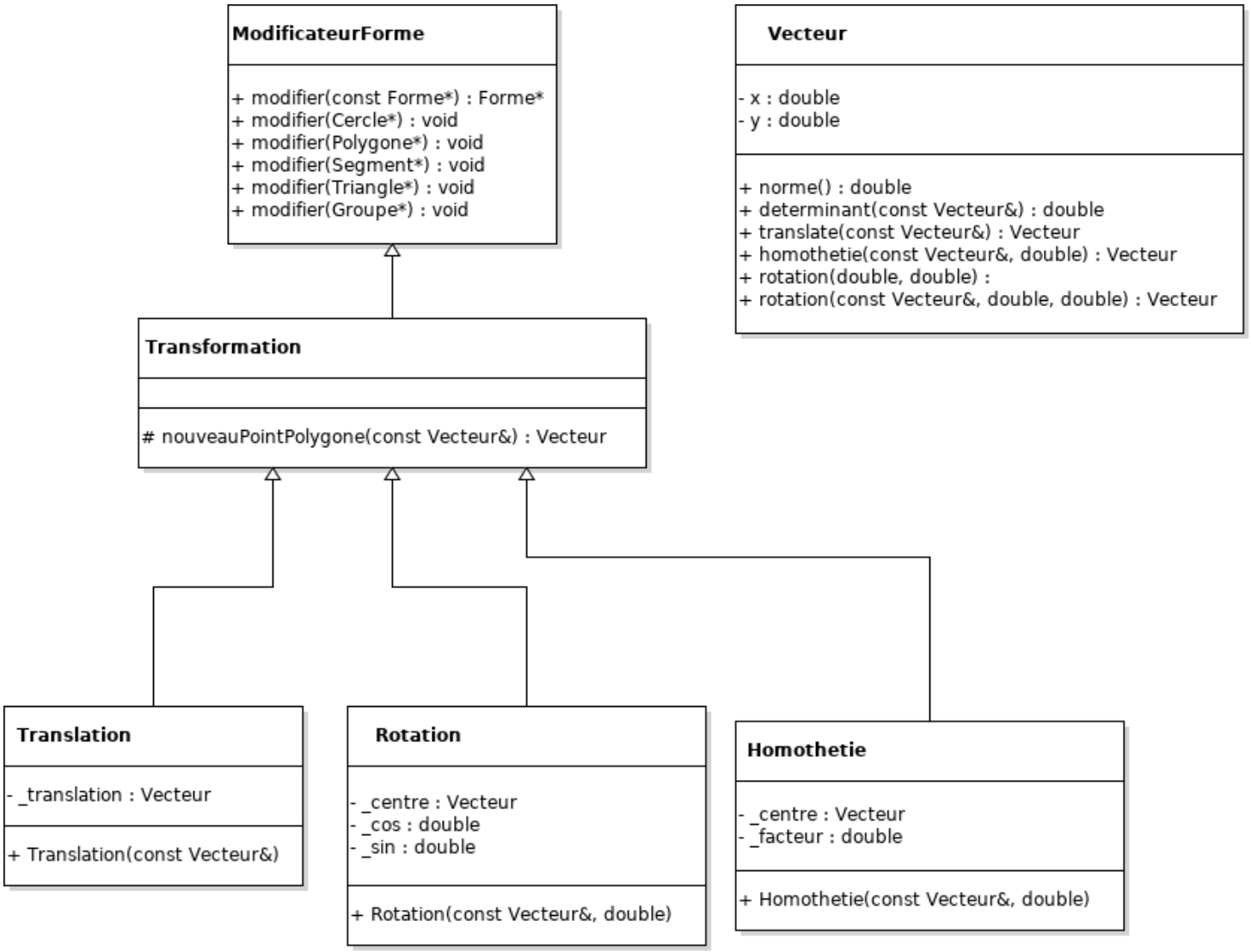
Chaque forme est représentée par la classe Forme, qui contient la couleur, le groupe et permet de calculer l'aire de la forme.

La classe Groupe hérite de Forme car un groupe de formes a aussi sa couleur, une aire et peut avoir un groupe parent.

Chaque forme dispose d'un pointeur vers son groupe (ou nullptr si elle n'en a pas), et chaque groupe possède une liste de pointeurs vers les formes qu'elle contient.

Le triangle n'hérite pas de Polygone pour ne pas hériter des méthodes permettant d'ajouter ou de supprimer des points, se retrouvant avec un objet invalide.

Transformations



Chaque transformation est représentée par une classe dérivée de Transformation.

Les transformations peuvent être appliquées de deux façons : directement sur la forme en utilisant Forme::modifier ou alors sur une nouvelle forme en utilisant Forme::modifierNouveau.

Le design pattern Visitor a été utilisé ici.

Les formules des transformations sont contenues dans Vecteur (qui représente les points), la transformation est appliquée à chaque point de chaque forme.

La rotation a été optimisée en calculant le cosinus et le sinus de l'angle directement dans la classe Rotation et réutilisant les valeurs pour chaque point (Vecteur).

Tests

Nous avons écrit des tests unitaires pour éviter les régressions. On utilise la bibliothèque catch.

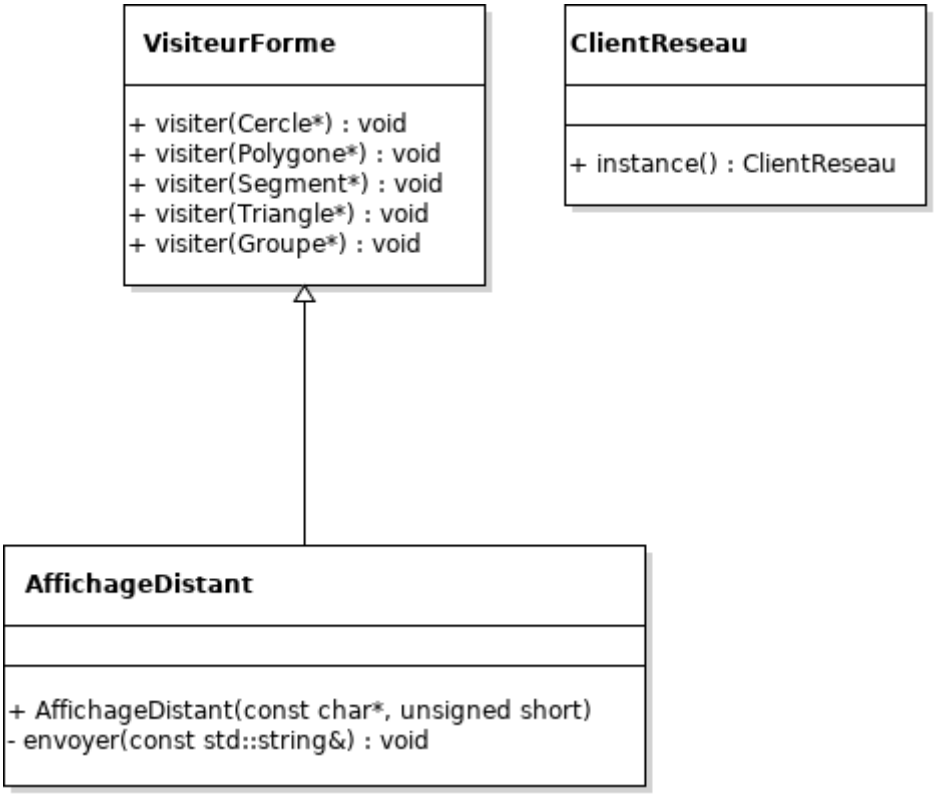
Pour chaque forme, on teste, les constructeurs par paramètres et par copie, l'opérateur = ainsi que l'aire.

Pour chaque transformation, on teste son application sur chaque type de forme.

Seuls l'affichage, la sauvegarde et l'opérateur std::ostream& << (std::ostream&) n'ont pu être testés.

Communication entre les deux parties

Transfert depuis le client C++



La classe ClientReseau est un singleton utilisé pour initialiser la bibliothèque Winsock sur Windows une seule fois au démarrage du programme et la libérer à la fermeture du programme.

La classe AffichageDistant est un visiteur de formes, qui va se connecter au serveur donné et lui envoyer toutes les formes avec le protocole détaillé ci-dessous.

Cette partie à du être écrite de façon à pouvoir utiliser le programme sur Windows et Linux, les directives de préprocesseur nous ont permis de créer un code portable. L'initialisation de la bibliothèque Winsock ne se fait que si le code est compilé sur Windows. Sur Linux, tous les types de Winsock qui n'existent pas ont été définis.

Protocole

La communication entre le client C++ et le serveur Java se fait avec un protocole basé sur les chaînes de caractères.

Le protocole envoie les formes sous forme de blocs, en commençant par le nom de la forme, suivie des variables (une par ligne) et en finissant par « FIN ».

```
SEGMENT
couleur=RED
debut=0;0
fin=100;100
FIN
```

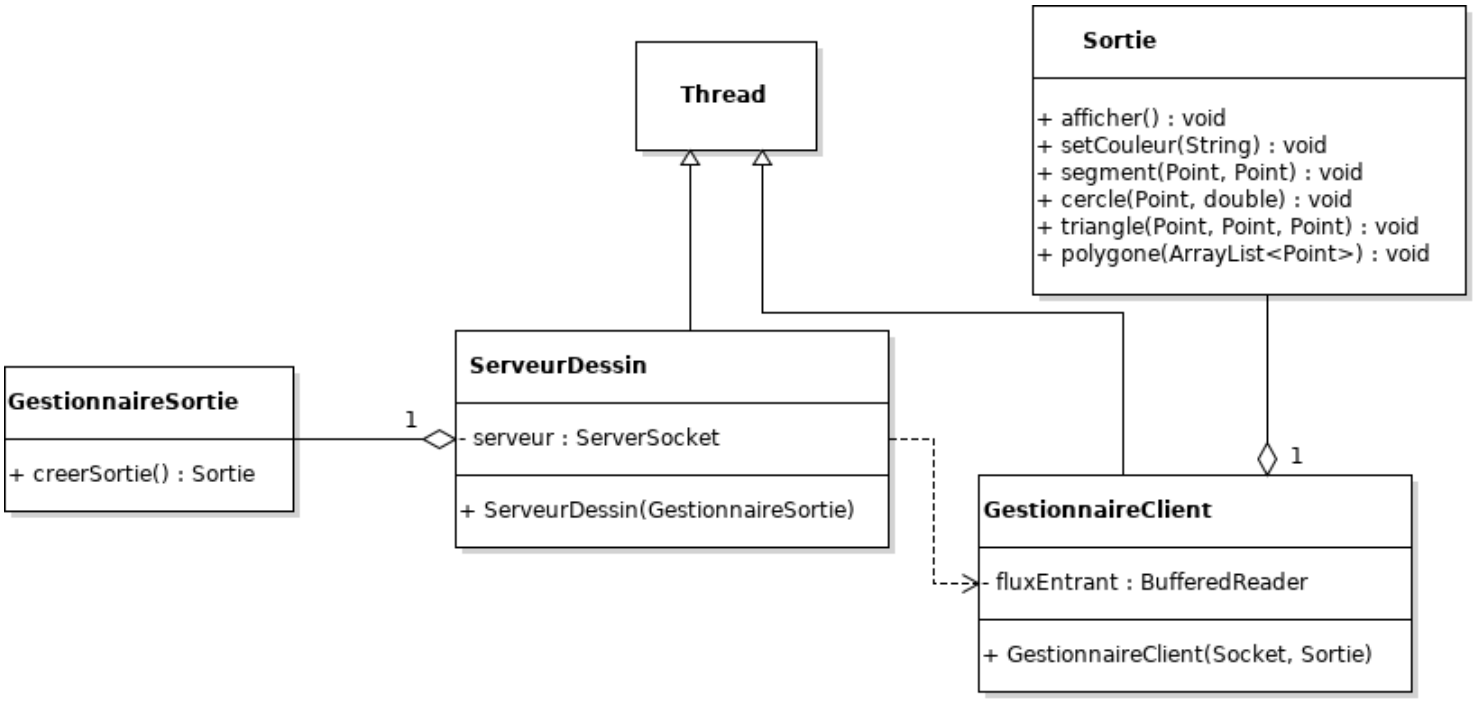
Forme	Variables
Cercle	• Couleur • Centre • Rayon
Polygone	• Couleur • Point (présent autant de fois qu'il y a de points)
Segment	• Couleur • Début • Fin
Triangle	• Couleur • Point 1 • Point 2 • Point 3
Groupe	Le groupe n'est pas envoyé, chaque forme est envoyée individuellement avec la couleur du groupe.

Cette forme à été choisie, car elle présente plusieurs avantages :

- On reçoit les variables une par une, sans devoir utiliser la méthode split.
- Le protocole est extensible, on peut écrire une nouvelle version de l'application avec de nouvelles variables (par exemple la couleur de la bordure de la forme), et garder une compatibilité avec la version précédente du serveur Java.

Les blocs se suivent, le bloc suivant se trouve sur une nouvelle ligne.

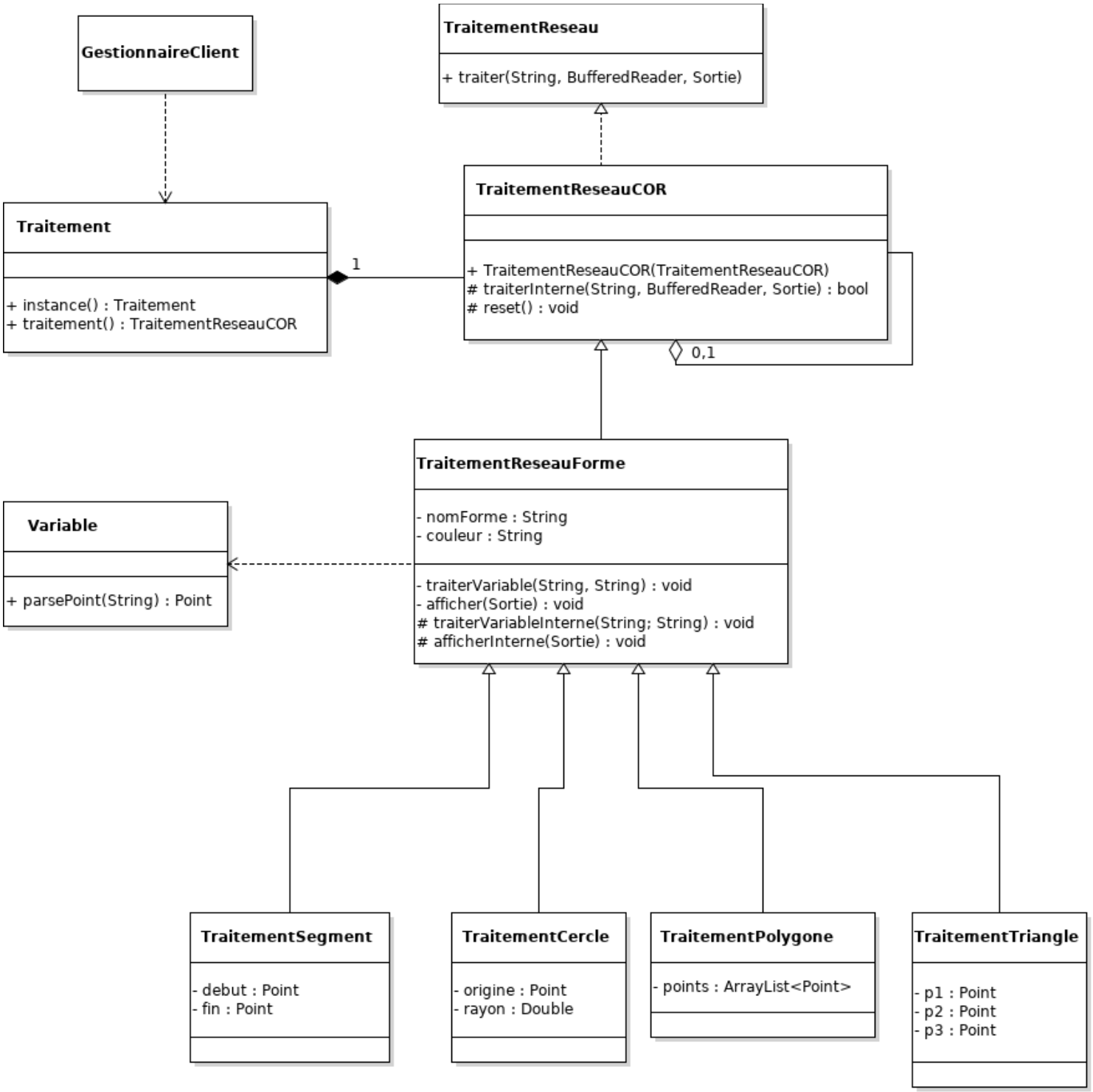
Réception depuis le serveur Java



La classe ServeurDessin s'occupe d'écouter sur un port spécifique, d'accepter les nouveaux clients, qui seront gérés par GestionnaireClient, et de leur attribuer une sortie (qui sera une fenêtre JavaFX ici).

La classe GestionnaireClient est lancée dans un nouveau thread pour ne pas bloquer les autres clients. La classe ServeurDessin l'est aussi pour ne pas bloquer le thread JavaFX.

Traitement des données reçues



Le serveur écrit en Java traite les données reçues à l'aide du design pattern Chain of Responsibility.

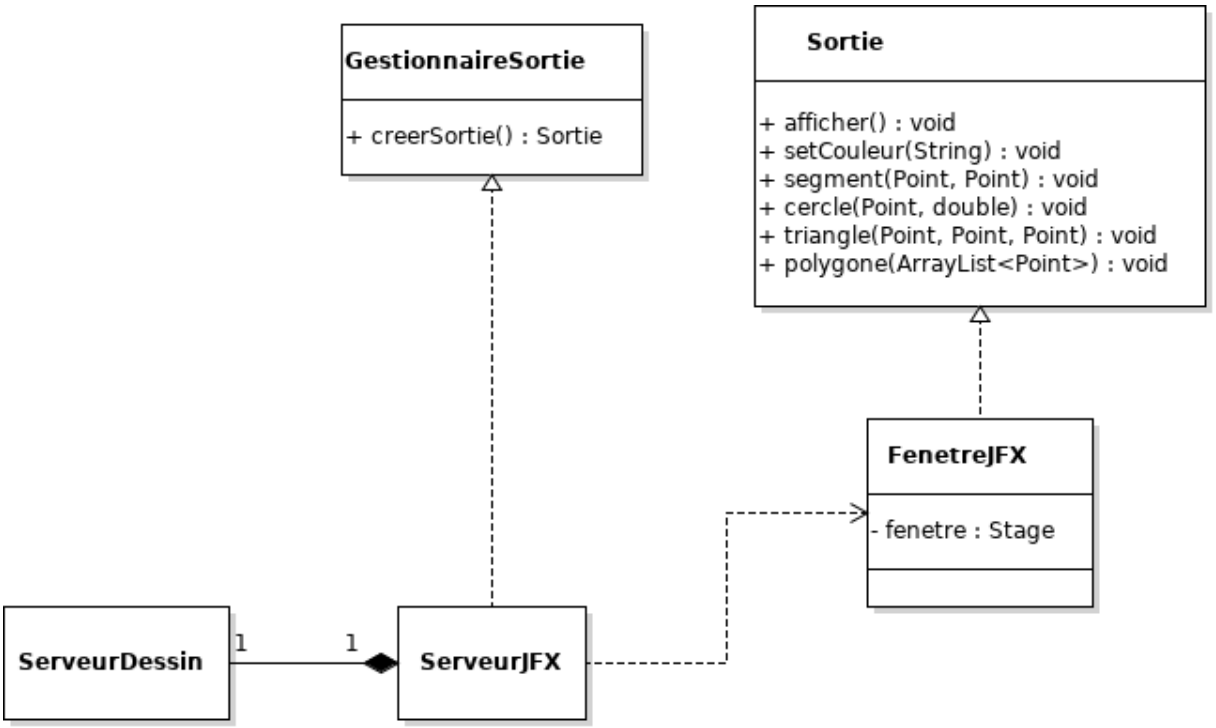
La classe Traitement est un singleton créant et permettant l'accès à la chaîne de traitement complète.

La chaîne est gérée par TraitementReseauCOR, qui décide si la classe peut traiter les données ou s'il faut les envoyer au prochain expert.

TraitementReseauForme s'occupe de traiter les variables reçues sous la forme nom=valeur. La classe dérivée s'occupera de traiter la valeur reçue comme chaîne de caractère, grâce aux méthodes statiques de la classe Variable.

À la fin du traitement d'une forme, l'expert est réinitialisé pour qu'il n'y ait pas les données d'une précédente forme qui apparaissent.

Affichage



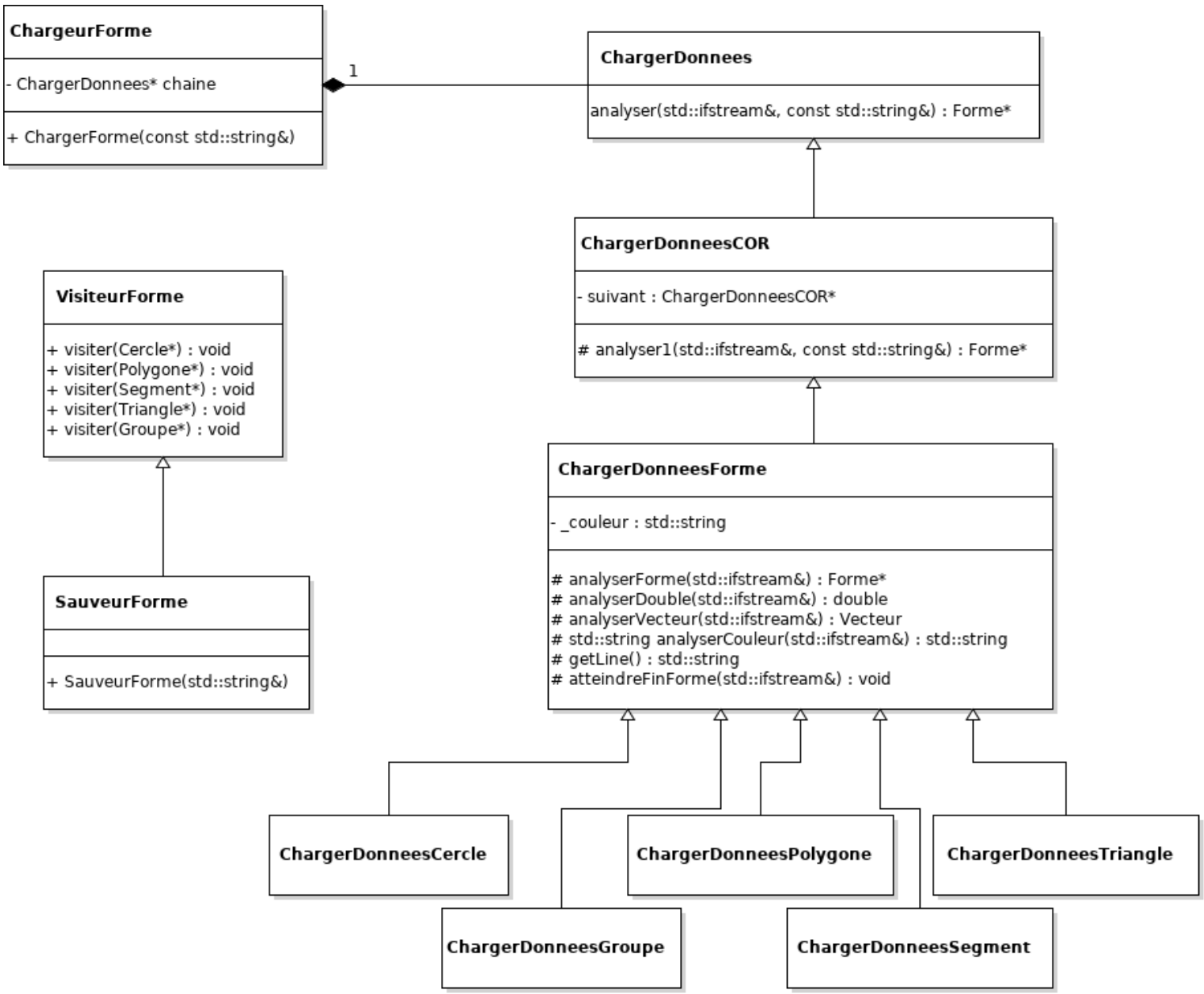
L'affichage se fait sur le serveur Java, qui utilise JavaFX.

Le programme lance directement le thread JavaFX, et le serveur écoute dans un autre port.

On utilise Platform.runLater() afin de pouvoir créer et afficher de nouvelles fenêtres depuis le thread qui gère le client.

La suppression implicite du thread JavaFX à du être désactivée afin de pouvoir garder le serveur fonctionnel indéfiniment.

Sauvegarde/chargement d'un fichier



La sauvegarde se fait de la même façon que la communication entre le client et le serveur.

Le code du transfert n'a pas été repris, car la sauvegarde doit supporter les groupes et les couleurs attribuées à la construction de la forme (qui prend la couleur du groupe à l'affichage) :

```
Cercle
red
FIN
357.000000,230.000000
20.000000
FIN
```

On utilise pas de noms de variables ici car seulement ce programme lira les fichiers exportés, on écrit donc les variables dans un ordre précis, le premier FIN permet de séparer la partie commune aux formes de la forme particulière. Le dernier marque la fin du bloc.

Cela permet de rajouter des variables et de pouvoir encore lire le fichier dans une version précédente du programme.

Le design pattern Visitor a été utilisé pour la sauvegarde et Chain of Responsibility pour le chargement.

La classe ChargeurForme crée la chaîne de responsabilité à la construction et charge le fichier passé en paramètre.

Le flux du fichier est passé à la chaîne d’experts qui analysera le nom de la forme et la traitera si l’expert la reconnaît.

Des fonctions ont été créés afin de repérer et lire des doubles, des Vecteurs ou des Couleurs. Le chargeur traite les formes les unes après les autres jusqu’à la fin du fichier.

Les formes chargées sont passées dans un membre privé vector<Forme*> de la classe ChargeurForme. Elles peuvent maintenant être consultées, modifiées ou envoyées au serveur Java.