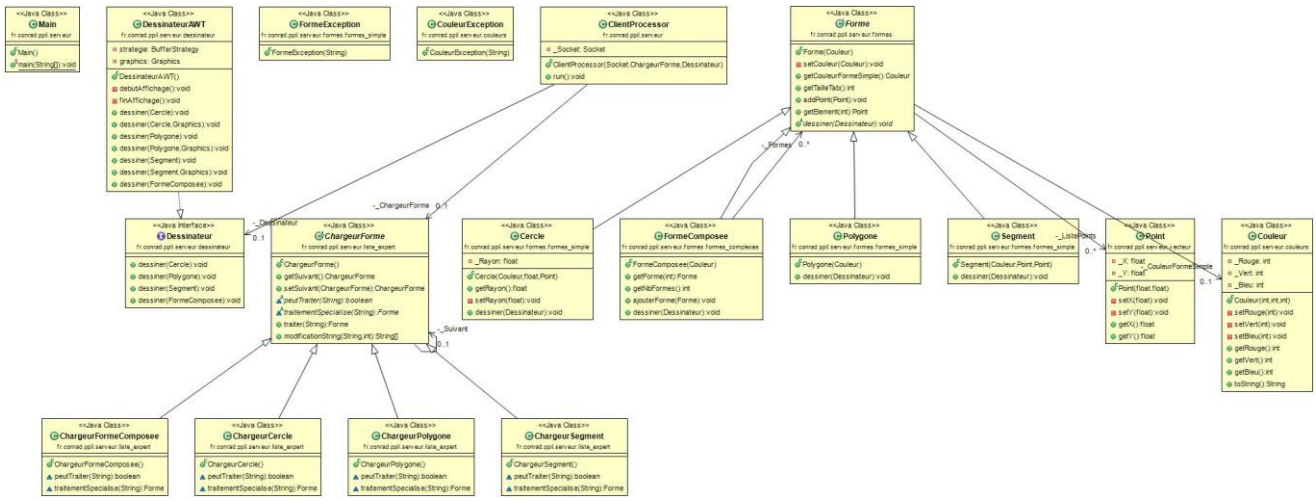
Serveur Java

Diagramme UML



Rapport

La partie java que nous avons dû réaliser consistait à faire le serveur.

Nous recevions une information par le client dans un format String d'une longueur inconnue que nous devions découper en plusieurs informations stockées dans un tableau et envoyées élément par élément à mon « Chargeur de Forme ».

Le « Chargeur de Forme » consiste à être une chaine de responsabilité qui est réalisée par un certain pattern. Cela nous permet de tester l'information par chaque fils et de faire un traitement spécialisé lorsque l'information reçue peut être utilisée par un des experts fils.

Les informations envoyées dans le « Chargeur de Forme » sont toujours sur la forme :

- Segment[(R,V,B), (x1,y1), (x2,y2)]
- Cercle[(R,V,B), (x1,y1), (rayon)]

- Polygone[(R,V,B), (x1,y1), (x2,y2), (x3,y3)]
- FormeComposee[Segment[(R,V,B), (x1,y1), (x2,y2)], Polygone[(R,V,B), (x1,y1), (x2,y2), (x3,y3)], Cercle[(R,V,B), (x1,y1), (rayon)]]

Le « Chargeur de forme » est initialisé au préalable dans « Main » et dispose de 4 experts. La liste des experts est la suivante.

- ChargeurSegment

  L'expert de segment est le premier à être appelé. Sa fonction « peutTraiter » consiste à inspecter le début du string pour voir si l'information correspond bien à « Segment ». Si c'est le cas son traitement spécialisé peut se lancer.

  Le « traitementSpecialise » de la chaine de responsabilité consiste à découper le String et retirer toutes les informations inutiles de mise en forme grâce une fonction « modificationString » qui nous permet d'être réutilisée par d'autre expert et ne pas avoir une redondance de code.

  Après ce découpage nous avons un tableau de String de 7 cases. Les 3 premières, comme pour chaque expert, correspond au code R,V,B qui nous permet de donner une couleur à notre forme.

  Les 4 suivantes correspondent aux deux points de notre segment, chaque point correspond à un x et un y.

  Nous avons donc réalisé pour plus de clarté une classe « Point » et une classe « Couleur »

  La classe « Point » nous permet de stocker les coordonnées d'un point directement dans une classe, qui dispose de « getter » pour chaque coordonnée.
  La classe « Couleur » consiste comme le point, elle nous permet de stocker les 3 niveaux de couleur dans une classe.

  Nous avons donc après découpage une classe couleur qui contient la « Couleur » reçu en entrée et deux classes « Points » qui disposent chacun d'un point du segment.

  Grâce à ces différentes informations nous appelons le constructeur de Segment avec la classe de couleur et les deux points crées.

  ```
  return new Segment(couleurSegment,p1,p2);
  ```

- ChargeurCercle

Le « ChargeurCercle » réalise les mêmes traitements que pour le segment, sauf qu'il test si l'information est bien un « Cercle ».

Un découpage du String est aussi réalisé dans le traitement spécialisé grâce à la fonction « modificationString ».

Les 3 premières cases du tableau de string est encore le code R,V ,B qui est envoyé dans le constructeur de la classe « Couleur »

Contrairement à segment ici nous avons qu'un seul point qui correspond à notre centre du cercle.

Et nous appelons le constructeur de la classe Cercle avec en argument la couleur, le point qui correspond au centre et la taille du rayon qui correspond à notre dernière information du tableau.

```
return new Cercle(couleurCercle, Float.parseFloat(temp[5]), centre);
```

- ChargeurPolygone

  Le « ChargeurPolygone » réalise au départ les mêmes traitements que « Segment » et « Cercle »

  Sauf que contrairement aux deux autres, un polygone peut comporter de trois (un triangle) à x points.

  Nous avons déjà appelé au constructeur d'un Polygone en lui ajoutant la couleur qu'on connait, car elle est toujours située dans les 3 premières cases du tableau.

  Nous avons donc réalisé un parcours du tableau qui commence au 3ème élément car avant les informations correspondent toujours au code de couleur. Le parcours s'incrémente par deux pour passer par chaque point. Et à chaque incrémentation nous créons un Points que nous ajoutons à notre polygone grâce à une procédure de la classe « Polygone »

  Et nous retournons le Polygone.

- ChargeurFomeComposee

  Pour la forme composée, le traitement est plus particulier. On découpe la chaine reçue et on l'envoie à nouveau dans une chaine de responsabilité qui réalisera de nouveau les informations vues au-dessus.

  Nous avons donc créé une liste de formes qui nous permettent de récupérer chaque forme reçue de notre boucle avec la chaine de responsabilité.

  Et de cette liste de forme on ajoute chacune des formes à notre forme composée qui nous permettra d'avoir sur une seule fenêtre toutes nos formes.

Nous avons aussi réalisé un « design pattern du visiteur ». Nous l'avons utilisé pour l'affichage.

Cela nous permet d'avoir un affichage pour chaque forme avec un constructeur différent. Mais aussi une meilleure portabilité car nous avons classe « DessinateurAWT » hérité de notre « Dessinateur », mais par la suite si nous devons afficher avec un autre moyen, nous avons juste à rajouter cette nouvelle classe.

Pour conclure, nous avons aussi réalisé une classe « ClientPorocessor » qui nous permet de réaliser du multi thread pour avoir plusieurs clients sur le même serveur.