Diagramme des classes

Le client

Les classes utilitaires et géométriques

La classe Vector2D Ils représentent des points ou des vecteurs. Ils sont la base de toutes les formes mais aussi de tous les vecteurs de transformations (translation, homothétie et rotation) qui seront réalisées.

Cette classe est composée de 2 nombres réèls pour chaques coordonnée x et y. Les opérations de bases (+, -, *) y sont implémentées grâce à la surcharge des opérateurs déja présents. Les opérations de transformations sont elles aussi présentent pour pouvoir les effectuer directement sur chaque point.

Les Couleurs Les couleurs sont gérées par une classe couleur constituée de 3 unsigned char : rouge, vert et bleu. Ce sont simplement les valeurs RGB qui vont nous permettrent de créer la couleur.

Elle dispose également d'une map contenant quelques couleurs de base déjà créées. La méthode qui permet d'accéder à la map de couleur est statique, il est ainsi possible d'y accéder sans que la classe soit instanciée. Les couleurs pré-enregistrée sont accessible grâce à une chaine de caractère caractérisant la couleur ("red", "blue" ...).

Les Angles Les angles, aussi ont une classe pour eux. Elle est très simple puisqu'elle se comporte comme une encapsulation d'un unique nombre réèl, la valeur en radian de l'angle. Son constructeur et ses méthodes de modifications (surtout des surcharges d'opérateurs) vérifient juste que l'angle est bien compris entre 0 et 2*pi, et l'ajuste (modulo 2*pi) si besoin.

Les nombreuses surcharges d'opérateur permettent d'utiliser les angles très simplement (+, -, *, cos, sin ...).

Les formes

La classe Shape La classe shape est la classe mère de toute les formes. Autrement dit, toutes les formes héritent de cette dernière. Le seul membre commun à toutes les formes est la couleur, qui est ainsi déclaré ici. Sans ce membre la classe pourrait se résumer à une interface. Et les méthodes sont nombreuses 1. Les méthodes de transformation géométrique et de calcul d'aire 2. Celles de dessin et de sauvergarde 3. Et les opérateurs << et (string)

Mis à part l'opérateur << et les getters/setters, toutes les méthodes sont des virtuelles pures. Nous reviendrons sur l'implémentation de ces dernières.

Chaque forme étant composée d'au moins un Vector2D, lorsque l'on applique une transformation sur une forme on effectue simplement cette transformation sur chacun de ses points, c'est à dire chaque Vector2D qui la compose. En effet l'opération est la même pour toutes les formes : pour appliquer un rotation à une forme, il faut appliquer une rotation à tous les points de la formes. Un seul cas particulier est à gérer, lors d'une homothétie avec un cercle ; il n'est pas possible d'appliquer une homothétie sur tous les points du cercle mais il suffit de l'appliquer sur le centre et de multiplier le diamètre par le ratio de l'homothétie.

Dans le cas de la forme composée, les transformations sont appliquées successivement pour chacunes des formes élémentaires qui la composent.

Les classe des formes Comme dit précedemment, les formes héritent toutes de la classe Shape. Elles sont chacunes composées de caractériques précises. La seule forme de sous-héritage est entre Triangle et Polygon puisque Triangle est aussi un Polygon.

Toutes disposent d'un opérateur de cast en string, qui vas nous être très utile puisque c'est celui la même qui va nous permettre de communiquer les caractéritiques de nos formes.

Les calculs d'aire Chaque forme a une fonction de calcul d'aire. Chaque forme a donc sa formule de calcul d'aire.

Pour le polygone :

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

Ou i est le ieme point du polygone et n le nombre de points.

Pour le triangle :

$A = \sqrt{p(p-a)(p-b)(p-c)} avec p = \frac{a+b+c}{2}$

Ou a, b, c sont les 3 côtés du triangle.

Pour le cercle :

$A = \pi * r^2$

Ou r est le rayon du cercle

Pour le segment l'aire est égale à 0.

Pour les formes composée l'aire est égale a la somme de toutes les formes qui la compose.

La classe de forme composée Les différentes formes la composent sont stockés dans un tableau de forme. Du coup chacune des opérations qui doivent être executées le sont pour chacunes des formes l'une après l'autre.

Le socket

Nous avons décidé de n'utiliser qu'un seul socket par client. Se socket ne doit donc être créé qu'une seule fois, c'est pourquoi il est implémenté grâce à un singleton.

Ce singleton mets à disposition plusieurs méthodes statiques que l'on peut donc appeler pour transmettre les messages au serveur. Le fait que ces méthodes soient statiques permet de les utiliser sans devoir tout le temps passer l'instance en paramètes.

Le socket possède 3 méthodes essentielles. La première permet de connecter le socket au serveur, en y renseignant l'adresse IP et le port de ce dernier. Les deux suivantes permettent d'envoyer et de recevoir, c'est à dire de communiquer avec le serveur. Les deux fonctionnent avec des chaine de caractères.

Une dernière fonction permet de fermer la connexion avec le serveur en fin de session.

Protocoles de communication avec le serveur Ce protocole doit permettre de faire passer la couleur de la forme ainsi que les informations permettant de la tracer.

Les différentes formes simples à tracer sont :

- Segment
- Cercle
- Triangle
- Polygone quelconque fermé (sans auto-intersection)

Le nombre de valeurs à faire passer dans les message varient en fonction des formes puisqu'elle n'ont pas le même nombres de membre. Et dans le cas du polygone il peut y avoir un nombre non défini de segments à tracer, selon la taille du polygone. La chaine de caractère qui va être envoyée au serveur commence donc par la nom de la forme ce qui lui permettra de savoir rapidement ce qu'il devra dessiner. S'en suivent ensuite la couleur sous sa forme rgb, puis en fonction de la forme, ses différentes caractéristiques.

Le protocole se comporte de la façon suivante :

`nomForme,red,green,blue,coord1,coord2,coord3,coord4,coord5,coord6`

Ce qui donnerait avec un exemple : `triangle,0,0,255,30,40,50,60,70,80` pour un triangle bleu avec (30;40), (50;60) et (70;80) en coordonées.

Dans le cas de la forme composée, la nom commencera par le mot `composedshape` puis les différentes formes y seront listées entre des '|'. Ce qui nous donne avec un exemple :

‘composedshape|triangle,r,g,b,1,2,3,4,5,6|segment,r,g,b,1,2,3,4

Chacunes des formes sera ainsi traitée une après l'autre.

Les visiteurs

2 classes visiteurs font partie du client. Une pour gérer le dessin avec le serveur, et une autre pour sauvegarder les différentes formes.

Le dessin des formes Le design pattern visitor est utilisé ici pour permettre l'implémentation rapide de nouvelles fonctionnalités de dessins.

Le visiteur de dessin est donc une classe abstraite permettant d'implémenter une méthode de dessin pour chaque type de formes.

Il faut ensuite de créer une classe descendant de cette interface pour implémenter différents modes de dessins, avec une classe par mode de dessin. Nous avons donc implémenté le dessin utilisant un serveur Java. Pour changer de mode de dessin il faudrait simplement ajouter une nouvelle classe héritant elle aussi de cette interface, qui pourrait alors contruire sa propre méthode de dessin, de manière totalement indépendante.

Dans ce cas, avec le serveur java, l'implémentation de chaque dessin est identique. C'est pourquoi l'interface DrawingVisitor fourni une fonction draw pour chaque type de forme mais également une générale pour un Shape. *Ainsi, si une méthode de dessin fait la même chose quelque soit la forme, il faut choisr d'implémenter la méthode draw prenant en paramètre un Shape.* Toutes les autres méthodes draw appeleront celle qui prend le Shape. *Sinon, si une méthode de dessin a un traitement différent pour chaque forme, on implémentera chaque méthode draw puis on laissera un corps vide pour draw(Shape).*

La méthode draw(Shape*) est la pour éviter la duplication de code dans chacune des méthodes.

Pour communiquer avec le serveur, la méthode draw utilise le socket ainsi que le protocole décrit auparavant.

Sauvegarde des formes La sauvegarde utilise un design pattern visitor pour les mêmes raisons que pour le dessin ; une nouvelle façon de sauvergarder peut se faire de mainère indepandante de la notre, et ce, relativement facilement.

La manière de sauvegarder les formes est très similaire à celle de les dessiner, puisqu'elle constiste elle aussi à récuperer la chaine de caractère de la forme puis de la traiter ; Ici elle est simplement écrite dans un fichier.

Le chargement des formes

Pour pouvoir charger des formes nous avons décider de créer deux chain of responsibility.

Une qui se chargera de reconnaître le format de sauvegarde utilisé.

Et l'autre qui se chargera de reconnaître la forme stockée et l'instanciera.

Ainsi la première chaîne reconnait le format de sauvegarde, traduit ce format dans le même format que l'on utilise pour transmettre les données au serveurs. On passe ensuite cette string à la chaine qui va retourner une instance de la forme qui était sauvegardée.

Dessin avec Qt (Fonctionnalité supplémentaire)

Nous avons choisi comme fonctionnalité supplémentaire d'implémenter un mode de dessin utilisant la librairie Qt.

Cela a demandé plusieurs modifications dans notre code. En effet lorsque l'on dessine avec Qt, nous utilisons une classe QtDrawer qui hérite de DrawingVisitor.

Cependant pour dessiner avec Qt nous devons créer une QGraphicsScene ainsi qu'une QGraphicsView qui vont changer lors des dessins.

Donc l'instance de QtDrawer ne peut être const. Comme ServerDrawer lui ne changeait rien nous avions passé le DrawingVisitor en const dans les méthodes draw des formes ce que nous avons donc du changer.

Le serveur Java

La deuxième grande partie de ce diagramme des classes est le serveur. Il est néanmoins beaucoup plus petit que le client.

Initialisation et fonctionnement

Une première classe gère le serveur en lui même, DrawingServeur. Il commence par démarrer, puis transmet toutes ses caractéristiques, qui sont l'adresse et le port de connexion, via la console. Une "boucle infinie" attend qu'un utilisateur se connecte puis lui alloue un DrawingThread.

Le drawingServer lui prend en charge un client. Il se charge de lui créer une fenêtre ainsi que d'initialiser les méthodes de dessins sur la fenêtre, à savoir la bufferStrategy et le graphics.

Ensuite une autre "boucle infinie" attend inlassablement les différentes requête de l'utilisateur, jusqu'à ce qu'il n'y en ai plus, ou que l'utilisateur ai envoyé le message "quit" au thread.

Les méthodes de dessin

Une chaîne de responsabilité vas se charger quant à elle d'analyser la requête de l'utilisateur pour pouvoir dessiner la forme correspondante.

La méthode draw va alors essayer tout les experts de dessins (cercle, segment...) jusqu'à trouver le bon (ou jeter une exception). Les différents experts split la chaine de caractère pour en extraire le premier mot (le nom de la forme), pour voir s'ils sont capable de la dessiner. Si la forme est reconnue le dessin peut être effectuer grâce a toutes les informations de la chaine de caractere.

Il est interresant de noter qu'il n'y a pas de drawer pour le triangle, puisque celui-ci est reconnu et traité comme un polygone par l'expert de polygone.