Rapport de laboratoire

**Département de génie logiciel et des TI**

|  |  |
| --- | --- |
|  | Double-cliquez sur les champs ci-dessous pour les modifier. L’entête de ce gabarit est normalement lié à ces informations et vous n’avez pas besoin de le changer. Pour effectuer les mises à jour dans l’entête, sélectionnez tout le texte dans l’entête et appuyez sur F9. |
| **No de laboratoire** | 1 |
| **Étudiant(s)** | Alexis Vuillaume |
| **Code(s) permanent(s)** | VUIA12119301 |
| **Cours** | LOG121 |
| **Session** | Hiver 2014 |
| **Groupe** | 1 |
| **Professeur(e)** | Christopher Fuhrman |
| **Chargé(e) de laboratoire** | Patrice Boucher |
| **Date** | 2014-02-07 |

Table des matières

[Introduction 2](#_Toc379496955)

[Analyse 3](#_Toc379496956)

[Conception 3](#_Toc379496957)

[Implémentation 6](#_Toc379496958)

[Discussion 8](#_Toc379496959)

[Manuel de l’utilisateur 9](#_Toc379496960)

[Conclusion 10](#_Toc379496961)

[Références 11](#_Toc379496962)

# 

# Introduction

Le but de ce laboratoire est de réaliser une application qui communique avec une application TCP/IP.

Le programme affichera des formes géométriques en fonction des informations que le serveur lui envoi. Pour recevoir une forme, l'application envoi "GET" au serveur.

Les informations envoyées par le serveur décrivent une des 5 formes possibles, un carré, un rectangle, un cercle, un ovale ou une ligne.

L'application doit interpréter ces informations pour afficher la forme correspondante.

A la fin de la discussion avec le serveur, l'application doit envoyer "END" au serveur afin de terminer la session correctement.

L'application devra être réaliser en suivant les principes de la programmation orientée objet (héritage, polymorphisme …), et les formes doivent être décrite comme étant des sous-classes de la classe Forme. Ces formes, lorsque l'ont souhaitera les afficher, seront stockée dans un tableau (array) uniquement.

Pour séparer la partie traitement de la partie communication, le SwingWorker sera utilisé afin d'établir la partie communication dans un thread à part et récupérer les informations envoyé par le serveur. Ces informations seront traité par une classe à part qui s'occupera de décomposer la chaîne de caractères à l'aide d'expression régulière.

Il existera aussi une classe qui s'occupera de gérer la génération des formes concrètes, afin que les instructions "new" ne soit que dans cette classe.

Enfin, pour développer cette application, un squelette fournit devra être utilisé. Celui-ci possède déjà un thread afin de s'occuper du traitement des informations provenant du serveur.

# Analyse

Lors de la réalisation de cette application, surtout 3 problématiques se sont dégagées,

La première partie concerne la partie communication avec le serveur, qui sera géré dans un thread à l'aide d'un SwingWorker.

La seconde porte sur le traitement des données reçues du serveur, qui devra être interprété au moyen d'expression régulière.

Enfin, la troisième sur l'aspect graphique de l'application, afin de représenter à l'écran les formes géométriques. C'est aussi pour cette partie qu'il faudra réfléchir aux choix d'implémentation des classes décrivant les objets, si toutes les classes héritent d'une même classe parente, ou si l'on considère qu'il y a des nuances que l'on peut apporter.

# Conception

La première partie à être implémenté est les classes décrivant les formes

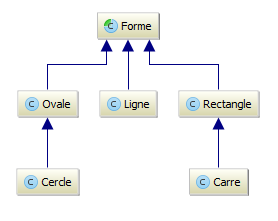
Pour cela, j'ai choisi de créer une classe Rectangle, Ligne, et Ovale qui héritent de la classe Forme. Ensuite, une classe Carré et Cercle qui héritent respectivement de Rectangle et Ovale, car un carré est un rectangle particulier, et un cercle, un ovale particulier.

Figure décrivant la hiérarchie des classes. Généré avec IntelliJ IDEA

Ensuite, il a fallu s'occuper de la classe permettant d'instancier des types de formes.

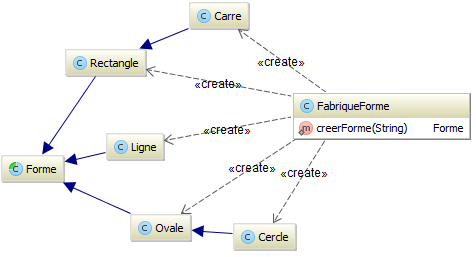
Pour cela, j'ai choisi d'utiliser le patron de conception "Fabrique" qui permet d'instancier des objets qui héritent d'une même classe abstraite [1]. La Classe Fabrique s'occupe d'instancier des formes grâce à la chaîne de caractère reçu, qui décrit la forme de façon détaillée.

Figure décrivant la classe FabriqueForme.

Généré avec Intellij

Pour interpréter cette chaîne de caractère, la classe FabriqueForme utilise la classe FormeParser, qui possède une méthode static "parse(String str)" qui permet de décortiquer la chaîne de caractère, et retourner un tableau regroupant les informations sur la forme. Grâce à ceci, FabriqueForme va être capable de discerner quel type de forme instancier, et quels sont ses attributs.



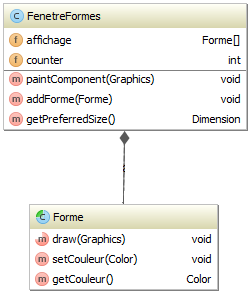
La classe chargée d'afficher ces formes est la classe "FenetreFormes", elle stocke dans un tableau au maximum 10 formes. Ces formes sont organisées de la plus récente à la plus ancienne, et au-delà de 10 formes dans le tableau, celui-ci remplace la plus ancienne forme par la nouvelle.

Figure décrivant la classe FenetreFormes

Généré avec Intellij

Le diagramme UML décrivant l'intégralité de l'application se trouve en annexe 1

# Implémentation

Classe FenetreFormes

Pour ajouter une nouvelle forme, j'ai choisi d'utiliser un tableau de forme de taille 10, lorsque celui-ci est complet, on remplace la plus ancienne forme par la nouvelle, selon la méthode "First In First Out"[2]

Variables : compteur // Contient l'emplacement du premier espace restant dans le tableau

Tableau<Forme> // Tableau de forme, de taille 10

Fonction ajouterForme(forme)

Si compteur > 0

Tableau[compteur] ← forme // On insère la nouvelle forme dans le tableau

compteur ← compteur -1 // On décrémente la dernière place disponible

Sinon

Pour i de 9 à 0 // On va parcourir tout le tableau pour avancer toutes les formes d'un rang, et faire redescendre la plus ancienne

Si i > 0

On permute Tableau[i] et Tableau[i-1]

Sinon

Tableau[i] ← forme // Si on est à la première case du tableau, on remplace la plus ancienne forme qu'on a redescendue par la nouvelle

Fin Si

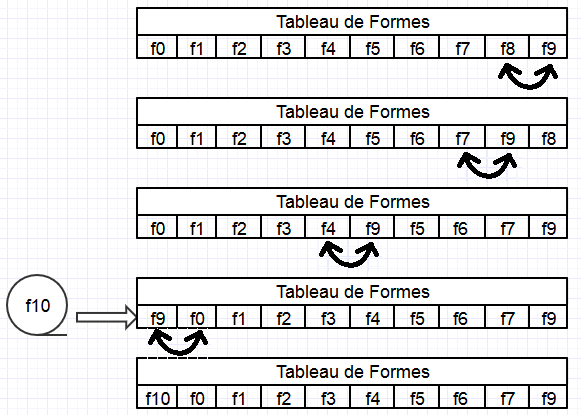
Fin Pour

Fin Si

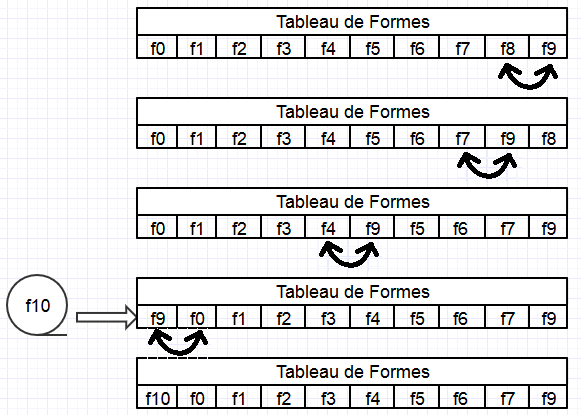
Fin Fonction

Voici le comportement du tableau :

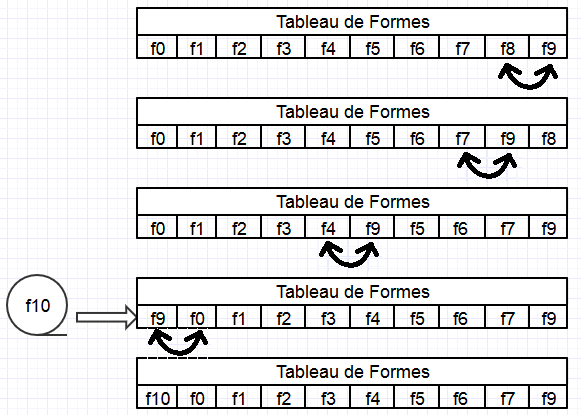
1re étape, on permute les éléments pour faire redescendre l'élément le plus ancien



Et on continue jusqu'à redescendre le plus ancien jusqu'à la première position



Une fois qu'on a fini de permuter tous les éléments, on insère la nouvelle forme à la place de la plus ancienne



Classe FormeParser

Dans la classe FormeParser, la méthode parse(String) permet de décortiquer la chaîne de caractère envoyé par le serveur. Pour cela, deux expressions régulières sont utilisées

La première est la suivante :

(?:<)[A-Z]\*(?:>)

Possède ">», mais ne fais pas partie de la capture

0 ou plus caractères entre "A" et "Z"

Possède "<», mais ne fait pas parti de la capture

La seconde expression régulière :

[0-9]{1,5}(?= )

Possède un espace, mais ne fait pas partie de la capture

Avec minimum 1 occurrence, maximum 5

N'importe quel caractère entre 0 et 9

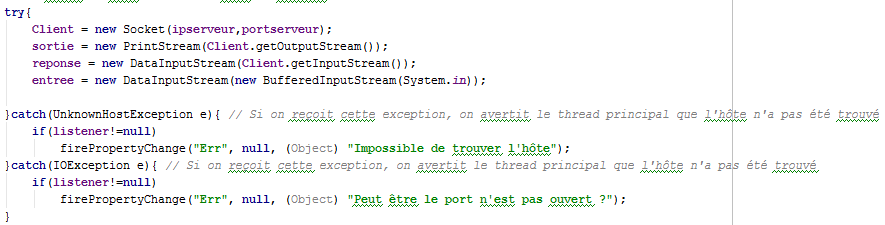
Classe FabriqueForme

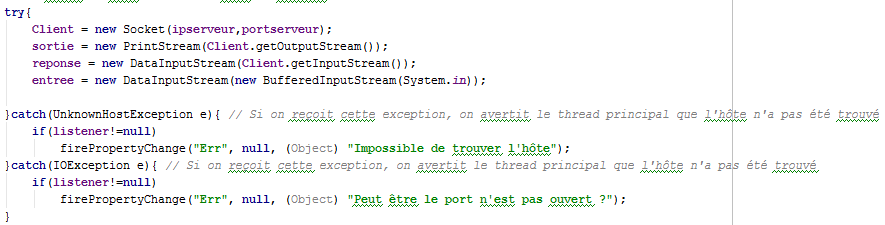
Cette classe possède une méthode statique creerForme qui permet d'instancier une forme en fonction de la chaîne de caractère passé en argument.

Cette chaîne de caractère sera parsée à l'aide de la classe ParserForme afin d'en extraire les informations décrivant la forme. En fonction du type de la forme reconnu, un rectangle, un carré, un ovale, un cercle ou une ligne sera instancié, et retourné

Classe Combase

C'est cette classe qui s'occupe de la communication avec le serveur grâce à l'utilisation des sockets. Celle-ci sert à communiquer entre deux hôtes via le protocole TCP/IP [3]

 Lors du lancement de la communication, on tente de lier le socket avec le serveur, de récupérer les flux de sorties et les flux d'entrées.

Ces actions sont entourées par les balises try/catch afin de capturer les exceptions si celle-ci est déclenchée, au cas où par exemple, l'hôte ne peut être trouvé, ou si celui-ci n'écoute pas le port sur lequel le socket tente de se lier.

En cas de levé d'exception, un listener est déclenché, afin de faire remonter l'information au thread principal.

La javadoc se trouve dans le fichier zip avec le code, dans le dossier "Javadoc TP1"

# Discussion

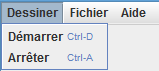
Pour l'implémentation des classes décrivant les objets, j'ai préféré faire hériter la classe carré de la classe Rectangle, et la classe Cercle de Ovale. De cette façon, on évite légèrement de la recopie de code, et l'ont pas facilement implanté de nouveaux comportements sans avoir à modifier chaque classe, mais uniquement les classes parentes. Il aurait été possible aussi de ne pas créer de classe Cercle et Carré, et simplement changer la couleur en fonction des paramètres d'entrées.

Pour le stockage des formes, j'ai préféré garder le tableau toujours trié, grâce à la permutation des éléments, mais il aurait aussi été possible d'utiliser une variable servant à pointer sur la case du tableau la plus ancienne (En utilisant un modulo). Sur de gros tableaux, cette technique aurait été plus rapide, mais sur un tableau de 10 éléments, le gain de temps d'exécution est négligeable.

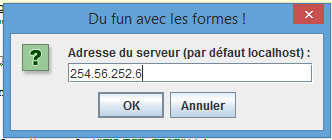
Une amélioration qui aurait été possible aurait été de placer le traitement des formes dans un autre thread, afin de rendre indépendant chaque partie du programme, un thread pour la communication, un pour le traitement des formes, et un pour les affichages.

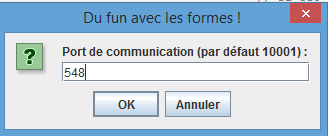
Une seconde amélioration possible aurait été d'afficher un pop-up pour prévenir lorsque la connexion au serveur échoue, et pourquoi

# Manuel de l’utilisateur

Pour compiler le .jar, il suffit de lancer dans un terminal la commande "java –jar application.jar

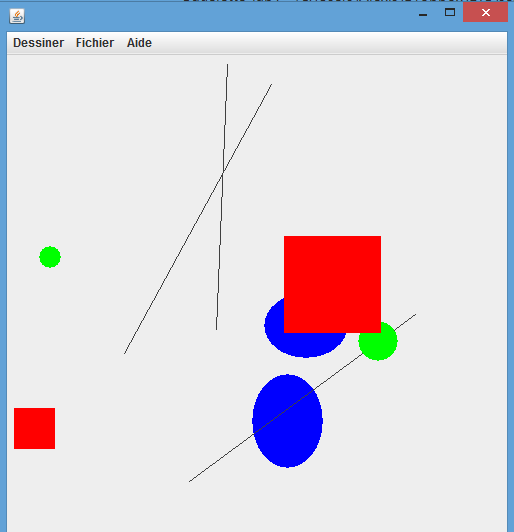
Une fois l'application démarrer, il suffit de cliquer sur démarrer dans fichier et d'ensuite indiquer l'adresse du serveur ainsi que le port dans les fenêtres qui apparaîtront afin que celle-ci se connecte via le protocole TCP/IP





Une fois connecté, l'application commencera à dessiner des formes à l'écran.

Pour arrêter l'affichage des formes, il suffit de retourner dans le menu "Dessiner" et de cliquer sur "Arrêter" ou alors d'utiliser le raccourcie "Ctr+A"



Il est possible d'arrêter l'exécution de l'application, soit en allant dans "Fichier" et "Quitter", ou alors de cliquer sur la croix rouge. Peut importe la solution choisis, l'application se fermera correctement après avoir envoyé la séquence de fin au serveur.

# Conclusion

Le but de ce laboratoire était de s'échauffer avec l'utilisation du langage de programmation Java, de mettre en oeuvre les bases de la programmation orientée objet, d'appréhender les notions de la communication client/serveur à l'aide des sockets en TCP/IP, mais aussi de développer un environnement graphique à l'aide de la bibliothèque swing de Java.

Cet exercice a aussi permis de mettre en évidence l'importance de séparer la partie communication et la partie affichage graphique.

Certaines améliorations peuvent être envisagées, comme par exemple séparer dans un autre thread la partie traitement des données envoyées par le serveur.

L'affichage des erreurs rencontré lors de la connexion à un serveur peuvent être elles aussi amélioré, par exemple en ajoutant un champ texte à la fenêtre qui afficherait l'état actuel du programme (Échec de connexion, connecté, en attente d'action de l'utilisateur, etc.)

Pour conclure, ce travail pratique m'a permis de rafraîchir mes connaissances en programmation Java, de remettre en pratique les paradigmes de programmation, et revoir les bases de la communication client/serveur par l'utilisation de socket.

# Références

[1] Article sur le patron de conception (Design Pattern) Fabrique sur Wikipédia :

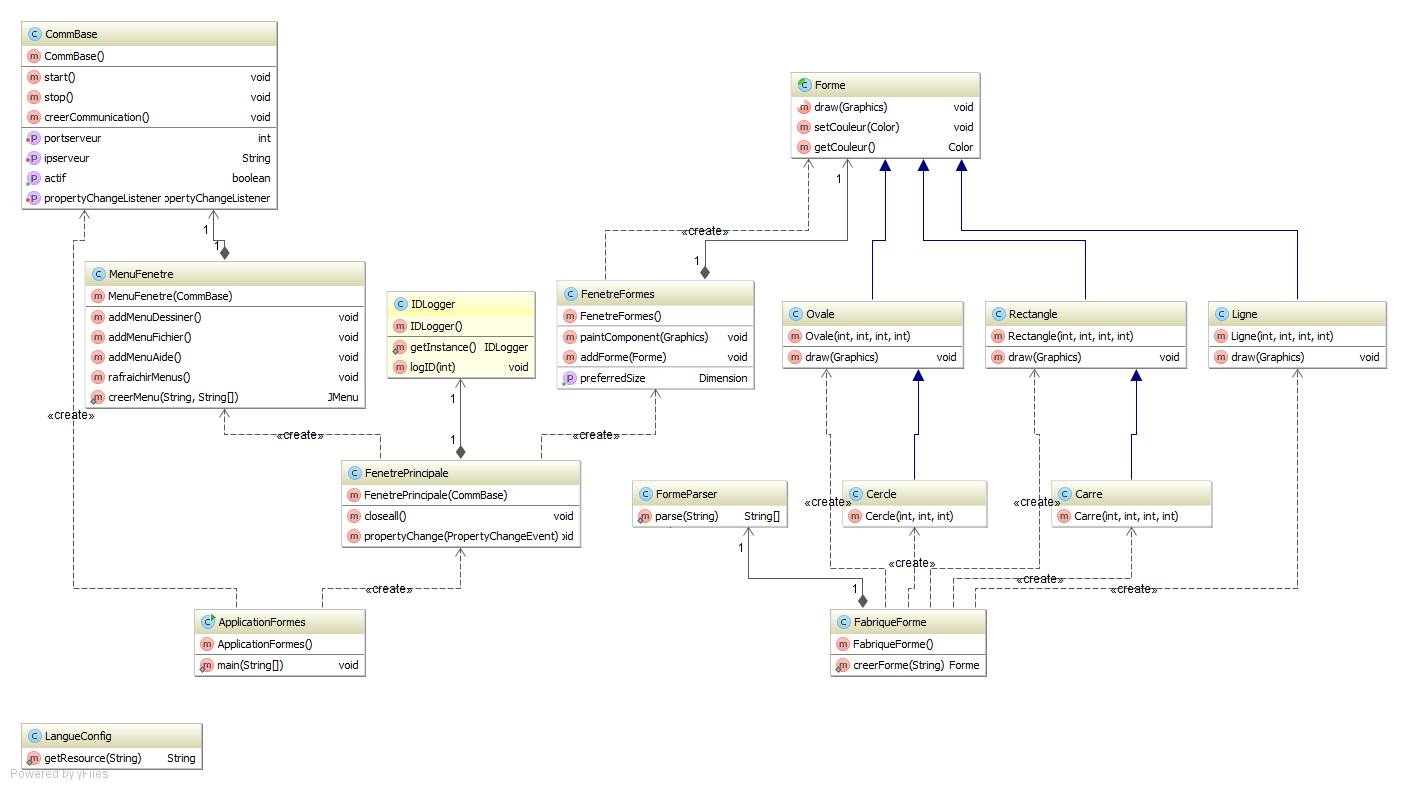
[*http://fr.wikipedia.org/wiki/Fabrique\_%28patron\_de\_conception%29*](http://fr.wikipedia.org/wiki/Fabrique_%28patron_de_conception%29)

[2] Article sur la méthode "FIFO" sur Wikipédia :

[*http://fr.wikipedia.org/wiki/Fifo*](http://fr.wikipedia.org/wiki/Fifo)

[3] Introduction aux sockets

[*http://fr.openclassrooms.com/informatique/cours/introduction-aux-sockets-1*](http://fr.openclassrooms.com/informatique/cours/introduction-aux-sockets-1)

*Annexe*