

Rapport de laboratoire

Département de génie logiciel et des TI

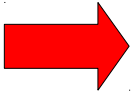
	<p>Double-cliquez sur les champs ci-dessous pour les modifier. L'entête de ce gabarit est normalement lié à ces informations et vous n'avez pas besoin de le changer. Pour effectuer les mises à jour dans l'entête, sélectionnez tout le texte dans l'entête et appuyez sur F9.</p>
N° de laboratoire	1
Étudiant(s)	Frédéric Bourdeau
Code(s) permanent(s)	BOUF10069403
Cours	LOG121
Session	Automne 2014
Groupe	3
Professeur(e)	Dominic St-Jacques
Chargé(e) de laboratoire	Alvine Boaye Belle Jean-Nicola Blanchet
Date	2014-09-292014-09-09

Table des matières

INTRODUCTION.....	3
ANALYSE.....	3
CONCEPTION.....	3
IMPLÉMENTATION.....	3
ALGORITHMES	3
DISCUSSION.....	3
MANUEL DE L'UTILISATEUR.....	3
CONCLUSION.....	3
RÉFÉRENCES.....	3

Introduction

Ce laboratoire consiste en la conception d'une application Java qui permet l'affichage de diverses formes géométriques à partir de données reçues par l'entremise d'un serveur.

Le tout doit être fait en respectant les normes de la programmation orientée objet. De plus, des concepts fondamentaux doivent être utilisés. Par exemple, pour la gestion des formes dans l'application, les notions d'héritage et de polymorphisme sont requises.

Le but de ce laboratoire est de réviser les principaux concepts de la programmation orientée objet, les bonnes pratiques. Il permet aussi de tester nos connaissances en programmation et en conception de logiciel.

Dans les sections suivantes, j'analyserai la situation proposée dans ce laboratoire, puis, je présenterai la solution que j'ai élaboré à l'aide de diagrammes UML. Ensuite, j'expliquerai comment s'est déroulée l'implémentation de la solution que j'ai conçue et je discuterai plus en détail des différents obstacles que j'ai rencontrés et des choix que j'ai du faire.

Suite à cela, j'expliquerai comment se servir du produit final à l'aide d'un manuel de l'utilisateur.

Analyse

Le serveur étant déjà fournis, l'application doit être en mesure de communiquer avec lui, c'est-à-dire, lui envoyer une des deux commandes "GET" et "END" et de recevoir sa réponse.

Cette réponse possède sa propre syntaxe, et doit donc être décodée à l'aide d'expressions régulières.

Une fois décodée, cette réponse doit être transformée en objet de type forme, selon ses spécifications.

La forme est ensuite ajoutée à une liste de formes qui s'affichent à l'écran.

Ce processus est répété tant que les deux applications (client et serveur) sont opérationnels et que l'utilisateur ne sélectionne pas l'option "Arrêter" du menu.

Le laboratoire comporte plusieurs défis :

- Communiquer avec un serveur par TCP/IP;
- Utiliser plusieurs fils d'exécution afin de séparer la communication avec le serveur et l'affichage des formes;
- Utiliser efficacement le polymorphisme;
- Toujours garder en tête que le code produit dans ce laboratoire sera réutilisé, et doit donc être facilement adaptable et réutilisable.

Conception

Afin de réaliser ce laboratoire, j'ai bien préparé la conception de l'application en créant un nouveau répertoire « [Git](#) » afin de grandement simplifier la gestion des sources. Par exemple, cela m'a permis de garder à jour mon projet simultanément sur mon ordinateur personnel et ceux du laboratoire informatique en quelques lignes de commande seulement.

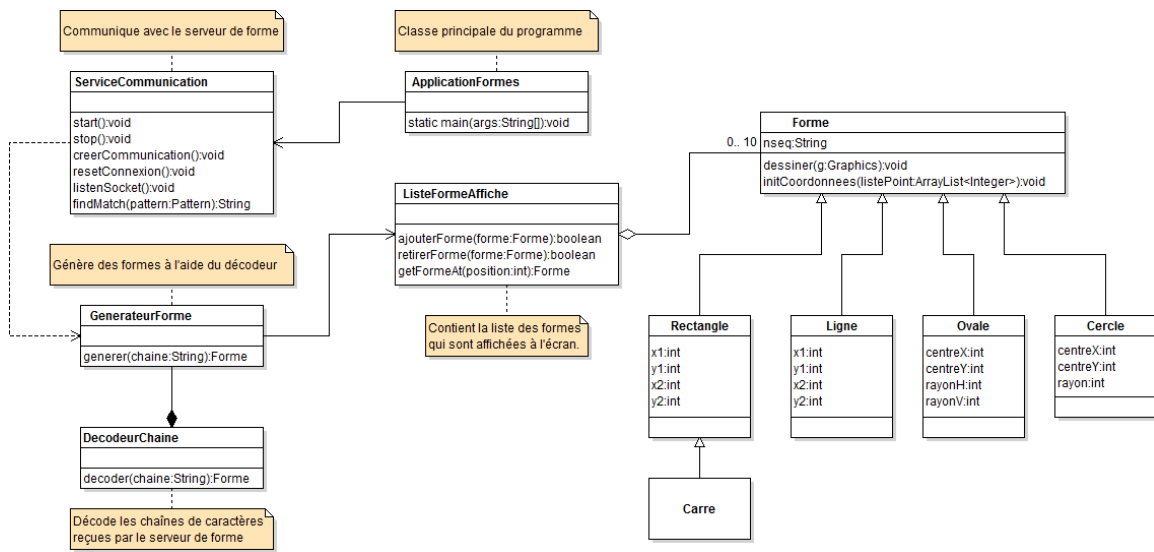
De plus, j'ai produit un diagramme de classe dans le but de bien planifier l'implémentation de ma solution (voir pages suivantes). Bien entendu, celui-ci a évolué au courant de la réalisation du laboratoire, dû à des obstacles non anticipés. C'est donc la version finale qui se retrouve dans ce rapport.

Les classes qui composent ma solution sont principalement celles du squelette donné avec l'énoncé que j'ai adaptées à mes besoins en plus de celles suggérées dans l'énoncé.

J'ai donc la classe principale de mon application qui, à son instanciation, construit la fenêtre principale, le service de communication et la liste de formes.

Le service de communication crée un second fil d'exécution dans lequel il communique avec le serveur de forme. Lorsqu'il reçoit une réponse, il l'envoie au générateur de forme, qui lui la retransmet à son décodeur de chaîne afin de générer la forme spécifiée.

Pour ce qui est de la structure des classes représentant les formes, j'ai créé une super classe nommée « Forme » dont les enfants sont chaque types de forme sauf le carré, qui lui hérite de « Rectangle », car un carré peut être représenté par un rectangle dont les côtés sont tous égaux.



Implémentation

Dans le but de rendre mon code le plus facile à maintenir que le pouvait, j'ai tenté de bien découper les diverses parties du laboratoire en classes ayant un but précis. Je me suis posé beaucoup de questions ressemblant à « Est-ce réutilisable? » ou « Est-ce vraiment en lien avec le but de la classe? ».

C'est donc cela qui m'a orienté dans mes choix. Le premier que j'ai du faire était au niveau de la structure de mes classes. Afin d'augmenter la clarté, j'ai répertorié mes classes dans des paquets (des sous-dossiers) avec des noms significatifs, comme « gui » pour les classes en lien avec l'interface ou « outils » pour les classes utilitaires comme la liste de formes, etc.

Une fois ma structure de paquets réalisée, j'ai créé ma classe « Forme » et ses sous-classes. À ce point, elle ne contenaient que les attributs relatifs à leur position dans la fenêtre. C'est suite à cela que j'ai réalisé qu'un carré pouvait être considéré par le programme comme un rectangle, et le cercle comme un ovale.

J'ai décidé de ne pas transformer ma classe « Cercle » en sous-classe de « Ovale », car je me serais retrouvé avec un attribut en trop. En effet, un ovale a un rayon horizontal et un rayon vertical, alors que pour un cercle, c'est la même valeur dans les deux cas. J'ai fait ce choix, car si jamais le programme devait avoir une validation plus poussée sur ses attributs, je devrais avoir une validation supplémentaire pour le cercle afin qu'il vérifie que les deux rayons sont toujours égaux.

Suite à cela, j'ai entamé l'adaptation de la classe « CommBase », que j'ai renommé « ServiceCommunication » par souci de clarté. En me fiant à la documentation et aux tutoriaux officiels (voir références), j'ai établie la connexion au serveur de forme grâce à la classe « Socket » de Java.

J'ai ensuite fourni un moyen à l'utilisateur de saisir l'adresse du serveur avec un « JOptionPane ». J'ai tenté de bien séparer la logique ici, car extraire le nom d'hôte et le port de la chaîne de caractères, tout en gérant les saisies invalides, nécessitait pas mal de lignes de code.

Une fois cette partie complétée, j'ai travaillé sur la génération de formes à partir de la réponse du serveur. J'ai donc adapté la classe « ServiceCommunication » pour qu'elle reçoive le générateur en paramètre à sa construction et qu'elle lui envoie cette réponse par l'entremise de la fonction « generer ».

Cette fonction reçoit la réponse et l'envoie au décodeur, qui est instancié en même temps que le générateur, grâce à la fonction « decoder ». Celle-ci décortique la réponse à l'aide d'expressions régulières que j'ai créé à l'aide d'un outil en ligne (voir références), puis, créé la forme avec les informations obtenues et renvoie la forme au générateur. Ce dernier l'ajoute à la liste (« ListeFormeAffiche ») qu'il a reçu au départ de l'exécution du programme.

La fenêtre qui affiche les formes (« FenetreFormes ») est avertie de l'ajout du forme dans la liste grâce au fait que la liste hérite de la classe « Observer » de Java et que la fenêtre implémente l'interface « Notifier ». La liste peut donc lancer un événement que la fenêtre est capable d'attraper. Ce lien est créé dans le constructeur de « FenetreFormes ».

Lorsque cette dernière trappe l'événement, elle enclenche la réévaluation de ce qu'elle affiche avec la méthode « repaint() » venant de base avec les composantes Swing. La méthode « paintComponant », qui dessine chaque forme de la liste, est ensuite appelée.

Cette routine est répétée à chaque fois que le serveur envoi une réponse.

Discussion

Les avantages de ma solution sont les suivants :

- Chaque classe est bien répertoriée. La structure générale est bien visible;
- Le projet est sur « git », ce qui facilite énormément la gestion de ses sources, donc le travail d'équipe;
- Le couplage des classes est limité. Il n'y a pas plus de trois relation entre chaque classe;
- Les classes ne comportent pas trop de lignes de code, à l'exception de la classe « ServiceCommunication », et sont documentées.

Pour ce qui est des désavantages, la liste est plus courte :

- La classe « ServiceCommunication » assez lourde et pourrait être réorganisée, voir même subdivisée en deux classes;
- Lorsqu'une saisie est invalide, il faut toujours re cliquer sur « Arrêter » dans le menu pour ensuite re cliquer sur « Démarrer ».

Manuel de l'utilisateur

Tout d'abord, afin que le programme fonctionne, il faut que Java 7 ou ses versions plus récentes soit installé.

Il faut aussi que le serveur de formes soit en exécution est accessible par TCP/IP.

Une fois le serveur fonctionnel, lancez le programme.

Ensuite, dans l'onglet « Dessiner », cliquez sur « Démarrer ».

Le programme vous demandera le nom d'hôte et le port. Entrez les sous la forme suivante : hôte:port.

Par exemple : localhost:10000.

Les formes devraient commencer à se dessiner de façon aléatoire dans la fenêtre.

Pour arrêter, toujours dans l'onglet « Dessiner », cliquez sur « Arrêter ».

Conclusion

Suite à cette vue d'ensemble sur la réalisation du laboratoire, je crois bien que j'ai répondu aux exigences, c'est-à-dire que la structure du programme respecte les normes de la programmation orientée objet et que toutes les fonctionnalités requises sont présentes. Il serait toutefois possible d'améliorer ma solution, surtout en révisant la classe de communication avec le serveur.

Grâce à ce laboratoire j'ai pu me réhabituer au Java et tester convenablement mes connaissances en orienté objet.

Références

Tutoriaux d'Oracle. *The Java™ Tutorials.*

<http://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>. Consultée le 24 septembre 2014.

Outil en ligne de test de Regex. *RegExr v2.0.* <http://www.regexr.com/>. Consultée le 24 septembre 2014.