Université du Québec
École de technologie supérieure

Département de génie logiciel et des TI

Rapport de laboratoire

N° de laboratoire	Laboratoire 1
Étudiant	Aissou Idriss
Code permanent	AISI01088901
Cours	LOG121
Session	Hiver 2014
Groupe	
Professeur	**Francis Cardinal**
Chargé de laboratoire	**Patrice Boucher**
Date de remise	6 Février 2014

1. INTRODUCTION (MAX 1/2 PAGE)

Le but de ce laboratoire est de mettre en place un client avec « Socket » permettant de se connecter à un serveur de formes .Par la suite, les formes géométriques reçues par le serveur devront être traduites puis affichées sur le panneau de dessin du client.

Dans ce laboratoire, différentes notions seront exploitées, telles que :

- ✓ Le concept de classe et de sous-classe ainsi que l'héritage
- ✓ L'utilisation des méthodes ainsi que l'interaction entre les objets
- ✓ Les diagrammes de classe en UML et leur mise en place
- ✓ Découper une chaine de caractères avec java.util.regex,

Puis la relation client-serveur

Une fois la réalisation faite, le projet permet de se connecter avec le serveur, de recevoir les formes sur le client, mais aussi de les afficher.

De plus, le projet donne l'opportunité de :

- ✓ Se connecter au serveur puis d'afficher les formes reçues
- ✓ Se déconnecter du serveur sans quitter l'application
- ✓ Gérer les erreurs concernant la spécification du nom de l'hôte et du port à joindre : erreur DNS, le serveur "X" ne répond pas sur le port "Y, le serveur s'est arrêté, timeout connexion,
- ✓ fermer convenablement la connexion lorsque l'utilisateur clique sur « X »

Au cours de ce rapport, nous verrons dans un premier temps la description de l'architecture logicielle, ainsi que le choix de la responsabilité des classes. Ensuite, nous aborderons les faiblesses de la conception. Enfin, nous évoquerons la décision de l'implémentation.

2. DESCRIPTION DE L'ARCHITECTURE LOGICIELLE

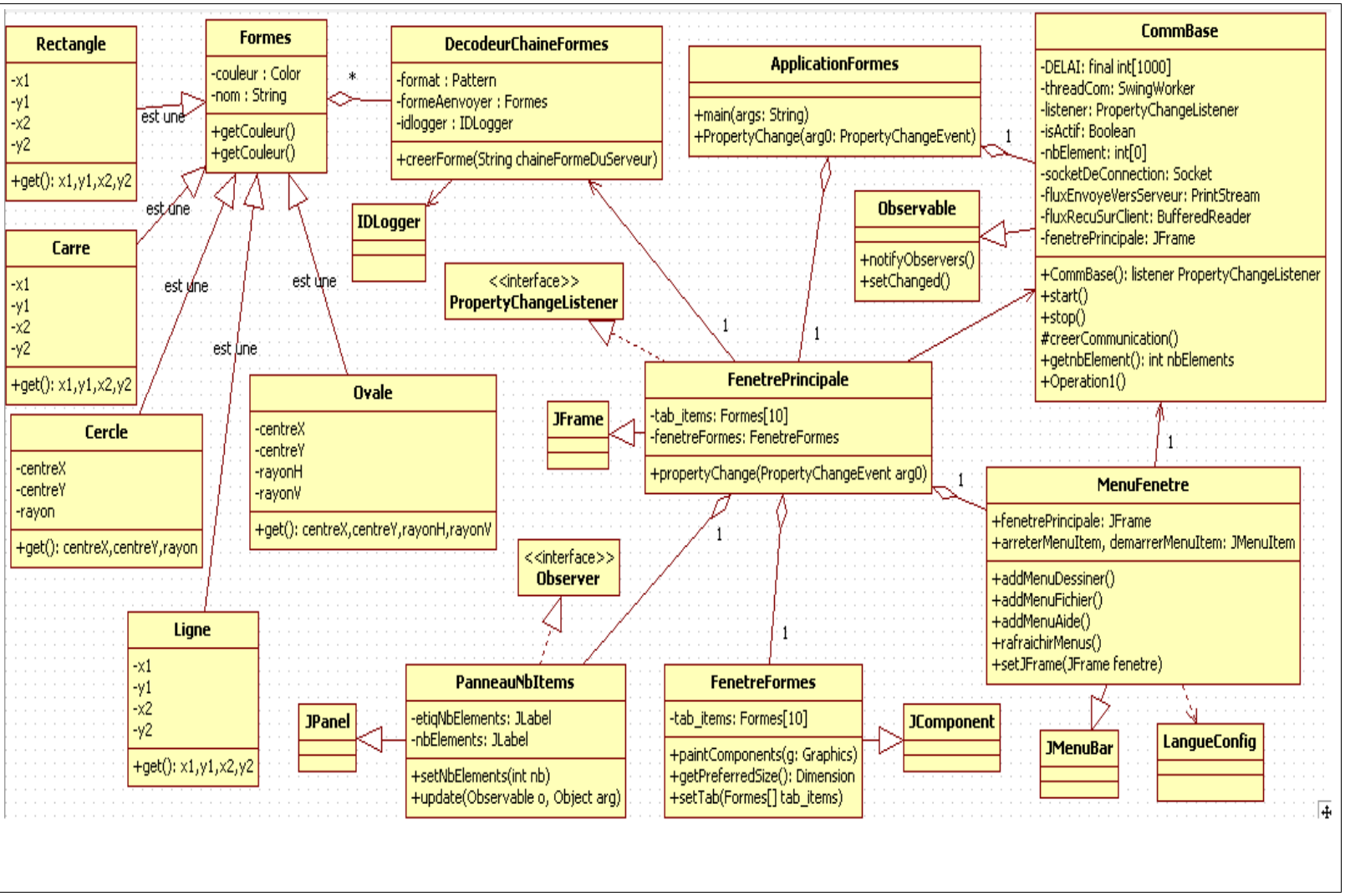
2.1. CHOIX ET RESPONSABILITÉS DES CLASSES

Classe	***Responsabilités***	***Dépendances***
« Formes »	– getCouleur() : permet de récupérer la couleur de la forme en question depuis la classe mère. – getNomForme() : Permet de récupérer le nom de la forme en question depuis la classe mère.	« DecodeurChaineFormes : c'est depuis cette classe que l'on utilise la classe Formes.
« ApplicationFormes »	– Main() : Permet de lancer l'application client.	–
« FenetrePrincipale »	– propertyChange() : permet de faire appel au décodeur de chaine dès la réception d'une nouvelle ligne de commande transmise par le serveur à la classe « CommBase ».	« ApplicationFormes » : C'est ApplicationFormes qui permet de lancer la Fenêtre Principale.
« FenetreFormes »	– paintComponents() : permet de dessiner les formes sur le panneau de dessin. – setTab() : permet de recevoir le nouveau tableau de formes dans la classe « FenetreFormes » afin de pouvoir les afficher.	« FenetrePrincipale C'est la fenêtre principale qui permet d'instancier Fenêtre formes.
« PanneauNbItems »	– setNbItems() : permet d'initialiser et de mettre à jour le « PanneauNbItems » selon le nombre de formes reçues, grâce à l'attribut « nbItems à « CommBase ».	« FenetrePrincipale » : C'est la fenêtre principale qui permet d'instancier PanneauNbItems ».
« MenuFenetre »	– Permet d'initialiser toute la structure du menu, les écouteurs, les boutons – addMenuDessiner()	« FenetrePrincipale » : C'est la fenêtre principale qui permet d'instancier

	– addMenuFichier() – addMenuAide()…	« MenuFenetre ».
« CommBase »	– start() : permet de créer la communication avec le serveur et de gérer les exceptions. – stop() : permet de stopper la connexion avec le serveur de manière sécuritaire. – creerCommunication() : permet de communiquer avec le serveur de formes en question, de recevoir les formes et de les transmettre la fenêtre principale.	« FenetrePrincipale » C'est la Fenêtre Principale qui contient le « MenuFenetre ». « MenuFenetre » C'est le « MenuFenêtre » qui permet de récupérer les identifiants de l'hôte puis d'utiliser la méthode start() de « CommBase ».
« DecodeurChaineFormes »	_ creerFormes() : permet de décoder la ligne de commande reçue, de créer la forme adéquate puis de la retourner.	« FenetrePrincipale » : C'est la fenêtre principale qui utilise le décodeur de chaine.

2.2 ARCHITECTURE LOGICIELLE

Diagramme fait avec StarUml.



2.3. DESCRIPTION DE L'ARCHITECTURE

Concernant l'architecture que nous avons choisi afin mettre en place notre application, nous avions comme optique de devoir diviser nos classe en différents sous-modules, de la façon suivante:

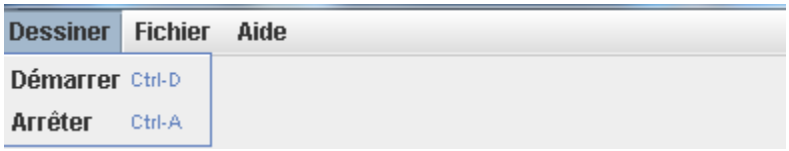
- Classe : « FenetrePrincipale »

  La fenêtre principale permet de recevoir les lignes de commandes reçues du serveur de la part de la classe « CommBase », ces lignes de commandes seront transférées au « DecodeurDeChaineFormes » qui lui retournera un objet de type Formes appropriée selon les caractéristiques de la chaine à décoder. Toutes ses formes seront ensuite ajoutées dans un tableau de formes [10].

  De plus, cette classe permet de permuter les éléments dans le tableau afin d'éliminer automatiquement la première forme et ainsi faire un roulement continu.

- Classe : « MenuFenetre »

  Cette classe permet d'initialiser le menu qui sera affecté à la fenêtre principale.



  Le menu Dessiner permet de gérer la connexion avec le serveur :

  Le bouton démarrer permet de se connecter au serveur et ainsi de lancer une boite de dialogue permettant de saisir les identifiants de l'hôte à joindre. Les informations récupérées seront envoyées à la classe CommBase.start() qui feront ensuite le traitement de la connexion.

  Le bouton arrêter permet de stopper la connexion avec le serveur de manière adéquate.

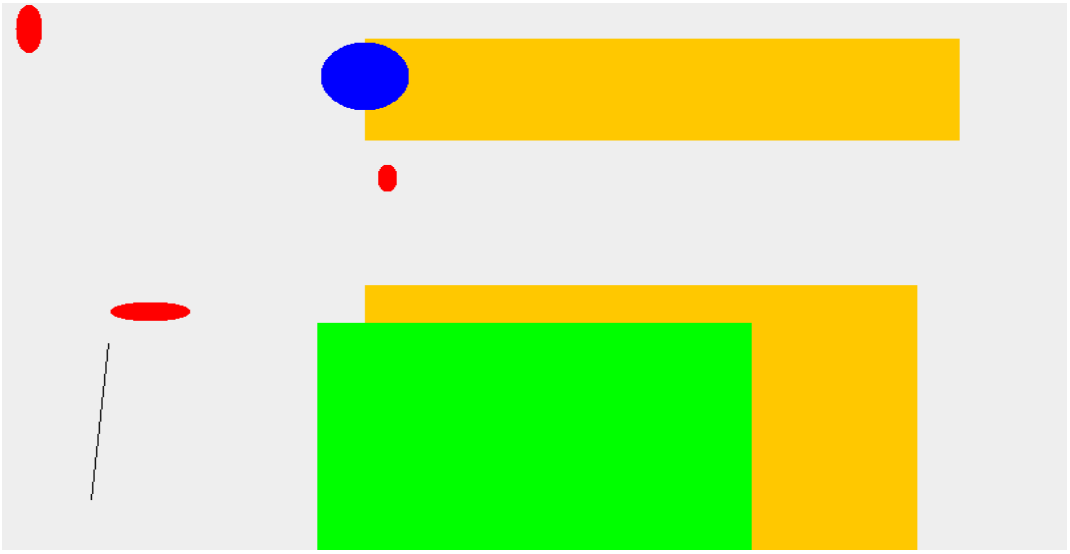
  Le menu fichier permet de quitter l'application.

  Le menu Aide permet d'afficher les informations sur le concepteur du code source.

- Classe : « FenetreFormes »

  Cette classe est appelée par la classe « FenetrePrincipale », permettant ainsi de pouvoir dessiner les formes grâce à l'utilisation de JComponent. La fenêtre principale envoie un tableau de formes dès la réception d'une nouvelle forme vers ce module pour ainsi afficher constamment 10 formes selon réception.

Explication graphique de « FenetreFormes » :



- Classe : « CommBase »

  Cette classe permet de créer la connexion avec le serveur tout en prenant en compte des différentes exceptions possibles.

- Classe : PanneauNbItems

  Permet d'ajouter un composant « JPanel » à la fenêtre principale ce qui a pour but de savoir le nombre total de formes reçues. Pour ce faire, la classe « CommBase » permet de lui retourner le nombre total de lignes de commandes reçues et ainsi actualiser le nombre de formes en temps réel.

  Explication graphique de PanneauNbItems :

- Classe : « DécodeurDeChaineFormes »

  Cette classe permet de créer la forme selon la chaine de caractère reçue. (Exemple ci-dessous)

  nseq <CARRE> **x1 y1 x2 y2** </CARRE>

  Dès la réception, de la chaine de caractère, le décodeur permet de décortiquer la chaine, de récupérer le nom de la forme puis de construire la forme avec les paramètres contenus dans la ligne de commande reçue. Dès lors, le décodeur retourne un objet de type forme à la Fenêtre principale.

  De plus, pour chaque forme reçue, le décodeur permet de décortiquer le numéro de séquence (nseq) et utiliser le classe « IDLogger » pour ajouter le numéro de séquence au journal, qui servira à son tour de fichier de validation.

- Classe : « Formes » (+ sous classe : Rectangle, Ovale, Carré, Ligne, Cercle)

  La classe forme, classe mère de Rectangle, Ovale, Carré, Ligne, permet de retenir le nom de la forme ainsi que la couleur qui définira le remplissage du dessin de la figure en question. Les sous-classes ont pour attributs leurs propres paramètres permettant le dessin de la figure sur le panneau dessinateur.

  Exemple : la classe Cercle aura pour attributs un rayon, un centre X et un centre Y.

2.4. FAIBLESSES DE LA CONCEPTION

Dans cette conception de l'application, il subsiste des faiblesses telles que :

La gestion de la connexion avec le serveur :

Ainsi, nous pouvons voir que la gestion de connexion avec le serveur s'effectue en deux parties. Premièrement, on récupère les identifiants à l'aide d'une boite de dialogue qui se déclenche dès que l'on clique sur le menu dessiner puis démarrer. Ensuite, ses identifiants sont transférés au module « CommBase » qui va se charger de gérer les exceptions si la connexion échoue, ou alors de créer la connexion. Nous pouvons voir que nous devons passer par deux classes différentes pour pouvoir gérer la communication avec le serveur.

Pour remédier à cette faiblesse, nous pouvons mettre en place une nouvelle classe « Connexion » qui gérera à elle seule la récupération des identifiants de l'hôte distant, puis de gérer la connexion avec le serveur afin de créer la communication et gérer les exceptions.

Le stockage des formes dans le tableau :

Au cours de cette application, les formes sont stockées dans un tableau « Array », la contrainte était de pouvoir afficher seulement dix figures à la fois. Cependant dans cette conception, dès que l'on reçoit une nouvelle forme, elle est ajoutée au début du tableau de formes. Cette insertion est possible tout en ayant décalé au préalable tout le tableau avec une boucle qui permet de parcourir le tableau de la fin vers le début, en indiquant que l'élément prendra la valeur de l'élément de l'indice – 1. Nous pouvons voir que cette méthode alourdit l'exécution du programme, car à chaque nouvelle forme reçue nous sommes obligés de parcourir le tableau du début à la fin. De plus, dès la réception d'une nouvelle forme, la fenêtre principale envoie un tableau complet de formes au panneau dessinateur de formes.

Pour remédier à cela, nous pouvons mettre en place une nouvelle méthode de stockage des données, telle que l'utilisation d'une file ou d'une « Arraylist » avec mise en place d'une classe Nœud pour alléger le traitement et le stockage des données. De plus, nous pouvons créer un système d'observer-Observable pour signaler une nouvelle forme au panneau dessinateur de Formes.

DécodeurDeChaineFormes :

Le décodeur de chaine utilise une méthode de découpage qui se focalise sur les espaces entre les segments de la ligne de commande. Par la suite, selon le segment numéro deux il compare cette chaine de caractères à des valeurs possible telles que : <\Ovale>.

Pour remédier à cette faiblesse, nous pouvons mettre en place une nouvelle méthode de découpage de chaine en récupérant directement le nom de la forme à créer dans la chaine. Ceci à pour but de créer la forme appropriée sans comparer à chaque fois une chaine de caractère a une autre pour en optimiser le temps d'exécution.

3. DÉCISIONS D'IMPLÉMENTATION

3.1. DÉCISION 1 : CONNEXION AVEC LE SERVEUR

- **Contexte**: gestion de la connexion avec le serveur

La solution choisie pour permettre de gérer la connexion avec le serveur et telle que nous affichons une boite de dialogue depuis le menu de l'application afin de pouvoir

récupérer les identifiants de l'hôte distant. Une fois les identifiants reçus, la méthode start () de la classe « CommBase » les reçoit et se charge de la création de la Socket, qui permettra de communiquer par la suite avec le serveur, mais aussi de gérer les erreurs de connexion.

Si la connexion échoue, des messages d'erreurs expliqueront la raison de l'échec (Warning message) grâce à la mise en place d'un bloc try{} catch {} qui permettra de gérer les exceptions susceptibles d'être levées.

Si la connexion réussie, alors un message indiquant que la connexion s'est établie apparaitra et la réception des formes débutera. La connexion s'établie à l'aide de la classe Socket.

Puis :

-Contexte: gestion de la communication avec le serveur

La solution choisie pour permettre de communiquer avec le serveur est de mettre en place une méthode créerCommunication() dans la classe « CommBase » qui permettra d'envoyer une commande « GET » au serveur et ainsi d'en recevoir la forme.

La forme est ensuite transmise à la Fenêtre Principale grâce à l'écouteur FirePropertyChange().

3.2. DÉCISION 2 : DÉCONNEXION DU SERVEUR DE MANIÈRE SÉCURITAIRE.

-Contexte: gestion de la déconnexion avec le serveur

La solution choisie pour permettre de se déconnecter du serveur sans quitter l'application est de déclencher une commande « END » dans la méthode stop() de la classe « CommBase » dès que l'on clique sur le menus Dessiner puis Arrêter. Ceci est possible grâce à la mise en place d'écouteur qui déclenche automatiquement la commande start() dans la classe « CommBase ». Ainsi, cela permet de pouvoir rester dans l'application même après l'arrêt de la communication avec le serveur.

-Contexte: gestion de la déconnexion de manière inappropriée avec le serveur

La solution choisie pour permettre de se déconnecter du serveur même si l'utilisateur clique sur le bouton « X » de la fenêtre est d'ajouter un WindowsListener() à la fenêtre principale dans la classe « FenetrePrincipale » qui va permettre de déclencher la

méthode stop() de la classe « CommBase » juste avant la fermeture de l'application.(windowClosing(WindowEvent arg0))

3.3. DÉCISION 3 : CRÉATION DU TABLEAU DE FORMES.

Contexte: Création des formes

La solution choisie pour créer les formes est de mettre en place un décodeur de chaine grâce à l'outil java.util.regex qui permettra de décortiquer la ligne de commande reçue du serveur et d'instancier la forme adéquate. Cette ligne de commande est envoyée au décodeur par la Fenêtre Principale. Ainsi, nous avons opté de séparer la chaine de caractère par ses espaces « » afin d'en extraire son nom. Une fois son nom extrait, on compte son nombre de caractères pour réduire le nombre de possibilités de comparaison et ensuite, on la compare à une autre chaine identique pour en découvrir la nature. Enfin, on en extrait les paramètres pour instancier la forme et la retourner à la Fenêtre Principale.

Contexte: IDlogger

La solution que nous avons choisie afin de mettre en place un journal sur les formes, est d'utiliser le « package » ca.etsmtl.log.util.jar dans la classe « DecodeurChaineFormes ». Ainsi, juste avant de retourner la forme crée dans la classe « DecodeurChaineFormes », nous faisons appel à l'IDlogger pour créer un journal avec le début de la ligne de commande reçue.

Contexte: Stockage des formes

La solution que nous avons choisie pour stocker les formes est d'utiliser un tableau [] « Array » depuis la classe Fenêtre Principale.

Contexte: Stockage constant de dix nouvelles formes

Pour pouvoir stocker toujours dix nouvelles formes dans notre tableau, nous avons d'utiliser une boucle « for » permettant de permuter les éléments de notre tableau de la fin jusqu'au début. Exemple (Pour de i=10 jusqu'a i=2 en algorithme)(tab[i]=tab[i-1)

Puis, nous insérons notre nouvelle forme au début du tableau.

3.4. DÉCISION 4 : AFFICHAGE DES FORMES.

Contexte: Affichage des formes

La solution adoptée pour afficher les formes sur le panneau dessinateur est de recevoir à chaque nouvelle forme reçue le tableau de formes dans la classe « FenetreFormes ». Grâce à la méthode setTab (Formes tab[]) de la classe « FenetreFormes », la Fenêtre Principale envoie au panneau dessinateur le nouveau tableau constitué des nouvelles formes.

Ainsi, la méthode paintComponents de « FenetreFormes » parcourt ce tableau et en affiche les composants.

4. CONCLUSION

Les objectifs de l'application étaient de pouvoir avoir un client capable de se connecter à un serveur, en utilisant un protocole TCP/IP sur le port 1000. Dès la connexion établie avec le serveur, le client reçoit des lignes de commandes(Formes) qui devront être décortiquées afin de pouvoir créer la forme géométrique adéquate sur le panneau de dessin. Cependant, seulement 10 formes peuvent être affichées sur la « FenetreFormes ».De plus, pour chaque forme reçue, le numéro de séquence (nseq) doit être ajouté à un journal qui servira de validation grâce à la classe IDLogger.

Ainsi en prenant en compte ces objectifs, notre application a permis de répondre à ces besoins.

La classe « MenuFenetre » permet de répondre au besoin de récupération du nom du serveur et le port TCP/IP de connexion.

La classe « CommBase » permet de répondre au besoin de gestion de la connexion. Grâce à cette classe, l'application donne l'opportunité de se connecter au serveur, gérer les erreurs de connexion (exceptions), de lancer la communication avec le serveur, mais aussi de pouvoir l'arrêter.

La classe « DecodeurChaineFormes »permet donc de répondre au besoin de découpage la chaîne de caractères reçue du serveur puis de créer les formes voulues. De plus, cette classe permet d'ajouter le numéro de séquence (nseq) à un journal, qui servira de validation grâce à la classe IDLogger.

La classe « FenetrePrincipale » permet de stocker les formes reçues dans un tableau [10] afin que la « FenetreForme » puisse les dessiner sur son panneau.

Au-delà du respect des contraintes, il subsiste quelques faiblesses dans la conception de l'application telles que :

La gestion de la connexion avec le serveur : nous pouvons voir que nous devons passer par deux classes différentes pour pouvoir gérer la communication avec le serveur.

Une solution telle que la mise en place d'une nouvelle classe « Connexion » qui gérera la récupération des identifiants du serveur, régira la connexion avec le serveur afin de créer la communication puis, gérera les exceptions pour permettre de solutionner la faiblesse.

Le stockage des formes dans le tableau : Nous pouvons voir que cette méthode alourdit l'exécution du programme, car à chaque nouvelle forme reçue nous sommes obligés de parcourir le tableau du début à la fin.

Pour pallier à cela, nous choisirons une nouvelle méthode de stockage des données telle que l'utilisation d'une file ou d'une « Arraylist » avec mise en place d'une classe Nœud pour alléger le traitement et le stockage des données. De plus, nous pouvons mettre en place un système d'observer-Observable pour signaler une nouvelle forme au panneau dessinateur de Formes.

En conclusion, ce laboratoire m'a permis de pouvoir prendre connaissance de nombreuses fonctions telles que :

- ✓ l'utilisation de la classe socket
- ✓ l'utilisation du décodeur de chaine regex
- ✓ la mise en place de diagramme iml

ce qui me permettra de les réutiliser dans des applications futures.

5. RÉFÉRENCES

http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
http://docs.oracle.com/