Client Formes

But du laboratoire

Le premier but de ce laboratoire, est de vous familiariser avec l'environnement de développement de logiciels de ce cours, soit Java. Vous devrez apprendre à utiliser la documentation du SDK (Software Development Kit) et à utiliser les différents outils de compilation et d'exécution de programmes disponibles au laboratoire.

Afin d'agrémenter le tout (!), ce laboratoire propose également d'intégrer différentes autres notions qui exploiteront davantage vos aptitudes de programmeur :

- programmation orientée-objet
  - concepts de classe, de sous-classe, de l'héritage
  - utilisation des méthodes pour faire interagir des objets
  - diagramme de classe en UML
  - rédiger et faire compiler un fichier .java source
  - exécuter une application Java
- client-serveur
  - notion de socket en TCP/IP, ouverture, lecture, écriture, fermeture
- contexte graphique
  - afficher/effacer une forme dans une fenêtre
- méthodes pour décortiquer une ligne de commande textuelle
  - Integer.parseInt, StringTokenizer, etc.
- ....

Description du laboratoire

Le laboratoire consiste à concevoir une application client, permettant d'afficher différentes formes géométriques primitives provenant d'un serveur.

Description du serveur

Le serveur que vous devrez utiliser, est une petite application TCP/IP qui répond uniquement à deux commandes :

- La commande **END** qui permet de terminer l'exécution du serveur.

- La commande **GET** qui permet d'obtenir une ligne textuelle qui comprend le nom d'une forme géométrique simple ainsi que les paramètres l'accompagnant.

Description d'une forme

La ligne provenant de la commande GET décrit l'une des formes suivantes :

nseq <CARRE> **x1 y1 x2 y2** </CARRE>
nseq <RECTANGLE> **x1 y1 x2 y2** </RECTANGLE>
nseq <CERCLE> **centreX centreY rayon** </CERCLE>
nseq <OVAL> **centreX centreY rayonH rayonV** </OVAL>
nseq <LIGNE> **x1 y1 x2 y2** </LIGNE>

Pour toutes les formes, **nseq** est un numéro de séquence codifié qui identifie la forme.

Pour le carré, le rectangle et la ligne, **x1 y1 x2 y2** représentent les coordonnées absolues des sommets par rapport à la fenêtre graphique.

Pour l'ellipse (ovale), **rayonH** représente le rayon horizontal
et **rayonV** représente le rayon vertical. Les axes des ellipses sont parallèles aux axes de l'écran.

Et bien sûr, **centreX** et **centreY** représentent la coordonnée (x,y) du centre d'un cercle ou d'une ellipse.

Fonctionnement manuel du serveur

Vous pouvez vérifier le bon fonctionnement du serveur de la façon suivante :

Si vous utilisez un serveur qui s'exécute sur une autre station que la vôtre :

Tapez : `telnet stationX 10000`

où le `X` dans `stationX` est le numéro de la station sur laquelle s'exécute le serveur.

Si vous utilisez un serveur qui s'exécute de façon locale :

Tapez : `telnet localhost 10000`

Une fois connecté, vous aurez le "prompt" suivant :

```
commande>
```

Si vous tapez **GET**, vous aurez une forme et ses paramètres tel que décrit plus haut. Si vous tapez **END** le serveur terminera son exécution. *Note : il y a un retour de chariot (\n) après le prompt.*

Spécifications de l'application client à réaliser

Vous devez réaliser une application client qui pourra se connecter au serveur, en utilisant un protocole TCP/IP (port 10000) et qui utilisera les deux commandes disponibles pour interagir avec ce dernier.

L'application client ouvrira une fenêtre graphique de 500 pixels par 500 pixels et affichera, une à une, les formes qui seront reçues par le serveur jusqu'à un maximum de 10 formes. Par la suite, la forme la plus ancienne doit être effacée pour permettre à une nouvelle forme de s'afficher. Une fois connecté au serveur, le client doit donc perpétuellement faire des requêtes de formes au serveur.

L'affichage des formes graphiques doit se faire comme dans l'exemple `SqueletteSwingApplication`, qui vous réalise déjà une partie de cette tâche.

Note : Si vous utilisez la classe `SqueletteSwingApplication`, vous n'avez qu'à dessiner la liste des formes dans la méthode `paintComponent` de la classe `Canvas`. Chaque fois que votre client reçoit une nouvelle forme, il demande au système de redessiner le `Canvas` avec la méthode `repaint()`. Ceci est la stratégie imposée par l'architecture d'interface Swing. Si vous faites autrement dans le cadre du `SqueletteSwingApplication`, vous risquez d'avoir des difficultés.

Les formes doivent être remplies et donc, ne doivent pas afficher que leurs contours. Il est conseillé d'utiliser une couleur unique pour chaque type de forme afin de bien discerner les formes semblables (carré vs rectangle).

Contraintes de conception

Le programme doit être fait suivant la programmation orientée-objet (pas procédurale). Si vous ne vous sentez pas bien avec les concepts de base en orienté-objet (classe, sous-classe, héritage), vous devrez contacter immédiatement un professeur avant de continuer avec ce laboratoire !

Chaque forme doit être décrite par une sous-classe différente de la classe-mère Forme, qui sera une classe abstraite. C'est-à-dire que l'utilisation de l'héritage est obligatoire.

La classe qui s'occupe de la communication doit être différente de la classe d'affichage. Normalement, si vous utilisez la structure proposée dans le squelette d'application Swing, la classe d'affichage sera utilisée à partir de la méthode `paintComponent()`.

Il doit y avoir une classe qui crée les formes concrètes, c'est-à-dire les instances des sous-classes de la classe Forme (Carré, Rectangle, etc.). Cette classe créatrice fournira une méthode qui prend comme paramètre d'entrée une String qui décrit la forme (c'est en fait la même chaîne de caractères qui provient du serveur). Cette méthode retournera un objet de type Forme, mais l'objet retourné sera une forme concrète (une instance de l'une des sous-classes). Le but est que toutes les instructions de **new** pour les sous-classes de forme (e.g., Carre, Rectangle, etc.) se trouvent uniquement dans cette classe.

Pour valider une partie du fonctionnement de votre application, vous devrez utiliser la classe IDLogger. Voici comment cette classe devrait être utilisée dans votre programme :

import ca.etsmtl.log.util. IDLogger;

…

IDLogger logger = IDLogger.getInstance(); //Méthode statique

...

logger.logID(id); //le id étant un int représentant le numéro de séquence reçu

Donc, pour chaque forme reçue, vous devrez décortiquer le numéro de séquence (nsec) et utiliser le classe IDLogger pour ajouter le nsec au journal, qui servira de fichier de validation."

Pour pouvoir utiliser IDLogger, vous devez informer Eclipse d'utiliser les classes qui se trouvent dans le fichier Java Archive (.jar) IDLogger.jar. Une façon d'y arriver consiste à copier le fichier IDLogger.jar dans le répertoire de votre projet, sélectionner votre projet dans la fenêtre "Package Explorer", sélectionner le menu "File", "Properties", sélectionner "Java Build Path" dans la liste des propriétés, choisir l'onglet "Libraries", cliquer sur le bouton "Add

External JARs...", sélectionner le fichier IDLogger.jar, fermer le dialogue "Properties for ..." en cliquant sur le bouton OK.

Pour utiliser IDLogger dans une console, vous devez également informer Java de l'existance de l'archive IDLogger.jar. Il suffit d'ajouter dans la variable d'environnement ClassPath la localisation de l'archive. Par exemple, si l'archive IDLogger.jar se trouve dans même répertoire que le reste de votre laboratoire, vous pouvez ajouter taper dans un shell Dos :

```
set ClassPath=%ClassPath%;.\IDLogger.jar
```

Ainsi lorsque vous lancerez l'une ou l'autre des commandes :

```
javac *.java
java monLab1
```

Java saura où trouver la classe IDLogger.

Rapport de laboratoire

Voir Style de correction et Rapports de laboratoire pour plus de détails.

Votre rapport doit comprendre un diagramme de classe en UML qui représente votre conception. Votre choix d'outil (Rational Rose, PowerPoint, Word, Visio, à la main, etc.) pour le diagramme n'est pas important.

Pondération

Il y a 12 points possibles :

- 6 pts = Fonctionnalité et présentation
- 6 pts = Rapport de laboratoire

Vous pouvez consulter la grille de correction.

Points boni

À faire seulement si vous avez réalisé toutes les fonctionnalités de base pour ce laboratoire. Vous pouvez gagner jusqu'à 1 point boni si votre client fournit à l'usager les options suivantes dans le menu d'application :

- Se connecter au serveur (et afficher ensuite les formes reçues), et

- Se déconnecter du serveur (cette option ne quitte pas l'application !)

Note : pour gagner les points boni, il faudra que l'usager puisse se connecter et se déconnecter plusieurs fois de suite sans aucun problème !

Nombre de séances

À confirmer lors de la première séance de laboratoire (environ 4 séances de 2h)

Date de remise et de présentation

À confirmer lors de la première séance de laboratoire. Voir la présentation du laboratoire.

Vous devez remettre le code source électroniquement au site Moodle. **Les remises par courriel seront refusées !**

Documents et fichiers utiles

Modèle Rational Rose UML -- à compléter, modèle UML en PDF.
Attention : le fichier MDL est censé être chargé dans l'outil Rational Rose disponible au local 3322.

StarUML -- Vous pouvez aussi utiliser cet outil gratuit pour concevoir vos diagrammes UML (Diagramme de classes, de collaboration et de séquence).
Cet application est également disponible au local A-3326

SqueletteApplicationSwing -- code source à réutiliser pour commencer.

IDLogger.jar -- à utiliser obligatoirement pour la validation de votre client (voir l'explication ci-haut).

ServeurForme.jar -- contient les fichiers .class pour exécuter le serveur de formes, à dézipper et à exécuter (en DOS) avec la commande "java -jar ServeurForme.jar".

Dernière mise à jour : 2018-01-09