Programmation répartie. Module 4102C
TP 4 : communications entre machines par le biais de sockets (suite) :
Client et serveur de messagerie instantanée.
année universitaire 2018-2019

Samuel Delepoulle et Frank Vandewiele

25 février 2019

Introduction

Le but de ce TP est réaliser une application de discussion en temps réel. Les clients communiquent avec un serveur qui aura pour tâche de répartir les messages à tous les clients connectés.

Première étape : connexion d'un client Swing au serveur

Récupérez la classe `ClientMessagerieV0` qui définit une interface Swing avec un client minimaliste (Une zone de texte, une zone de saisie et deux boutons).

Vérifiez que vous pouvez compiler ce client. Attention, il est défini dans un package, vous devez obligatoirement vous placer à l'extérieur du package pour compiler et exécuter :

```
javac client/ClientMessagerieV0.java
java client.ClientMessagerieV0
```

Travail à réaliser : la connexion / déconnexion

Pendant le TP, vous pourrez tester votre programme sur le serveur qui vous sera indiqué. Le serveur, après avoir accepté vos connexions, réalisera

un simple écho (il vous retournera votre message) jusqu'à recevoir la chaîne "quit".

- L'action sur le bouton "Connexion" doit . . .connecter. Pour cela, créez la socket correspondant à l'hôte indiqué dans la zone de saisie.
- Définissez les flux d'entrée et de sortie de la socket.
- Un message indique le succès ou l'échec de la connexion.
- En cas de connexion, le bouton "connexion" devient "déconnexion".
- L'action sur "déconnexion" devra déconnecter.
- Si le client n'est pas connecté à un serveur, désactivez le bouton "envoi".

Travail à réaliser : l'envoi de message et la réception de la réponse

- Lorsque le bouton "envoi" est cliqué, le message contenu dans la zone de saisie est envoyé au serveur et sa réponse est affichée dans la zone de texte.
- Les éventuelles exceptions sont affichées dans la zone de texte accompagnées d'un message adéquat.

Messagerie : le serveur

Vous allez maintenant réaliser la partie serveur. Pour cela, vous allez commencer par utiliser le serveur du TP précédent.

Pour rappel, voici comment il fonctionne :

La classe Service

C'est la classe qui réalise la connexion entre un client et le serveur

- Elle hérite de la classe `Thread` pour pouvoir s'executer en tâche de fond pour chaque client.
- Elle comporte un attribut de la classe `Socket` et un constructeur qui permet d'initialiser cet attribut.
- Dans la méthode `run()` :
  - Après avoir initialisé les flux d'entrée et de sortie,
  - prévoyez une boucle de lecture-écriture : lorsqu'une chaîne est lu, elle est recopiée sur le flux de sortie.
  - jusqu'à ce que l'utilisateur saisisse la chaîne "quit".

La classe ServeurMultiThread

- C'est la classe principale de l'application
- Dans la méthode `main`, Créez une instance de `ServerSocket`
- Dans une boucle sans fin :
  - Placez le serveur en attente (méthode `accept`)
  - Lorsqu'un client se présente, on crée une nouvelle socket qu'on utilise pour créer et démarrer une instance de `Service`.

A ce stade, vous devriez pouvoir tester votre serveur avec votre client. Faites également tester ce serveur par d'autres étudiants et/ou l'enseignant de TP.

Client qui réalise des lectures et écritures asynchrone

Avant de permettre au serveur de diffuser les messages à tous les clients, il va être nécessaire de modifier un peu le client.

L'objectif étant de réaliser une application de messagerie, les clients ne doivent pas réaliser une boucle écriture puis lecture séquentielle. Il faut dissocier les processus de lecture et d'écriture. L'écriture vers le serveur sera toujours déclenchée par l'action sur la zone de texte mais c'est maintenant le serveur qui va envoyer les messages et le client les lira au fur et à mesure.

Pour cela :

- Le client doit maintenant implémenter `Runnable` (ceci est obligatoire puisqu'il hérite de `JFrame`, il ne peut pas hértiter de `Thread`).
- La partie lecture doit être déplacée vers la méthode `run()` qui comportera une boucle de lecture. Lorsqu'une ligne est lue, elle est recopiée dans la zone de texte.
- pensez à démarrer un `Thread` lorsque qu'un client est connecté à un serveur.

Votre client est donc maintenant prêt à se connecté à la version du serveur de diffusion. Réalisez le test de connexion. A ce stade, vous devriez être en mesure de "discuter" si vous êtes connecté à un serveur de diffusion.

Le serveur de diffusion

Votre serveur va maintenant réaliser la diffusion des messages envoyés par un client vers tous les autres. Pour cela, vous allez utiliser le patron de

conception "observateur" (voir le modèle des classes dans le cours du module 3105).

Voici comment procéder :

— Ecrivez les interfaces `Observable` et `Observer`.
— Ecrivez la classe `Diffusion` qui implémente `Observable`. Elle comporte un attribut (`String`) qui contient le message envoyé à tous. Lorsque cet attribut est modifié par la méthode `setMessage()`, tous les abonnés sont notifiés.
— La classe `Service`, elle va implémenter l'interface `Observer` :
— Elle contient un attribut `static` de la classe `Diffusion` qui sera initialisé au premier lancement d'un `Service`. Pour cela, testez si sa valeur est différente de `null`.
— Chaque client sera ajouté aux observateurs.
— Lorsqu'un message est lu, on appelle la méthode `setMessage()` de la classe diffusion qui appelle la méthode `update()` de tous les `Service`.
— Déplacez l'écriture du message dans la méthode `update`.

A ce stade, votre serveur est capable de diffuser un message reçu à l'ensemble de ses clients connectés. Vérifiez que votre serveur réalise correctement cette opération.

Améliorations

Il est possible de réaliser beaucoup d'améliorations au niveau du serveur :

— donner un nom à chaque client de façon à identifier qui parle ;
— gérer la liste des utilisateurs connectés ;
— transmettre des messages privés (envoyé depuis un utilisateur à un autre).
— donner la possibilité d'envoyer des images, des fichiers.