INSA

B3125
Nicolas GRIPONT
Rim EL IDRISSI MOKDAD

COMPTE RENDU TP

Etat d'avancement du TP : fini.

Partie I : RMI

Le but du TP est de se familiariser avec la notion de RMI en créant une fenêtre de chat où un client pourra envoyer des messages à tous les autres clients du serveur. Il pourra aussi recevoir les messages des autres clients.

L'utilisation d'un RMI registry consiste à créer un Registry dans la classe server avec un port de connexion bien défini. On crée un stub de ServerInterface pour pouvoir utiliser les services de cette classe à distance. Ensuite, on connecte le stub au registre à l'endroit destiné au server de nom « server » à l'aide de la fonction bind(). Pour faire le lien serveur→client, ce dernier doit avoir un attribut de type ServerInterface. Comme on a crée un stub de ce type on pourra alors, côté client récupérer l'objet ServerInterface présent dans le registre. Les classes que nous avons implémenté géreront alors l'envoi et la réception des objets de type MessageInterface en créant, à chaque fois, des stubs de ces objets.

Nous avons implémenté une interface graphique qui permet au client de rentrer un nom d'utilisateur afin de se connecter au serveur. Si le nom d'utilisateur choisi est utilisé par un autre client qui est déjà connecté au serveur, la connexion est refusée jusqu'à ce que le client rentre un nom d'utilisateur valide.

Une fois le client connecté, l'historique des anciens messages lui est affiché sur son interface ainsi que la liste des clients actuellement connectés au serveur.

Lorsque le client envoi un message, celui ci est affiché sur l'interface de tous les clients connectés ainsi que sur sa propre fenêtre de chat.

L'enregistrement de l'historique se fait quand on éteint le server dans un fichier XML. Au lancement du serveur, l'objet Server charge les messages enregistrés dans le fichier XML.

Partie II : Socket

Le but du TP est de se familiariser avec l'utilisation des sockets comme moyen de communication entre le client et le serveur. On recrée la fenêtre de chat précédente.

La particularité ici revient à l'utilisation de threads : un thread client (la classe

ClientThread.java) afin de pouvoir récupérer ce qui est envoyé via une socket côté serveur à une socket côté client ; un thread server (la classe ServerMultiThread.java) qui va, avec une socket server, attendre la connexion d'un client et créer une socket de communication et un autre thread server (la classe ServerThread.java) pour communiquer avec le client. Enfin le ServerThread va écouter les messages envoyés par le client via la socket client.

Nous avons implémenté la même interface que précédemment avec les mêmes contraintes sur le nom d'utilisateur. Quand le client se connecte, la liste des clients actuellement connectés au serveur s'affiche mais l'historique n'est pas régénéré.

Lorsque le client veut envoyer un message, il peut choisir de l'envoyer à tout le monde ou à un client en particulier. Il suffit que le client sélectionne le nom du client dans la liste avant d'envoyer le message pour que celui-ci s'affiche uniquement sur son interface et sur celle du client ciblé. Lorsque le client sélectionne « all » avant d'envoyer son message, celui-ci est affiché sur l'interface de tous les clients actuellement connectés au serveur ainsi que sur sa propre interface.