TP 1
Programmation distribuée avec sockets

3 heures

1 Un objet serveur

La première étape consiste à écrire un objet contenant une socket permettant d'accepter des communications distantes. Pour cela, on veut pouvoir accepter les communications sur un port spécifié en paramètre sur la ligne de commande.

1. Créez une classe nommée *Server* ayant les attributs et constructeurs nécessaires (n'instanciez pas encore de sockets) ;
2. Ajoutez une méthode *main* permettant de créer un objet *Server* sur le numéro de port passé sur la ligne de commande (une application robuste doit vérifier le nombre et le type des paramètres) ;
3. Créez une méthode *execute* qui :
   (a) instancie une socket serveur ;
   (b) attend une connexion d'un client ;
   (c) affiche sur la sortie standard un message indiquant quand une connexion est acceptée.

2 Premier test

Modifier votre serveur pour que son comportement soit de base l'exécution de la méthode *execute*.

1. Démarrez le sur votre machine locale ;
2. Dans un terminal, utilisez la commande *netstat* ou *lsof* afin de vérifier que votre serveur est bien en attente de connexion sur le port que vous avez spécifié ;
3. Simulez un client à l'aide de la commande *telnet*. Connectez vous à votre serveur et vérifiez son bon fonctionnement.

3 Un client

Communiquer avec le serveur grâce à *telnet* n'est pas très pratique. C'est pourquoi nous allons écrire un programme Java le faisant pour nous.

1. Créez le squelette d'une classe nommée *Client* permettant de se connecter à un serveur dont le nom et le port sont passés en paramètre ;

2. Ajoutez à cette classe une méthode *execute* qui ouvre une socket vers le serveur ;

3. Vérifiez que votre client fonctionne correctement.

4 Transmission d'information

Une socket possède 2 streams, un pour la lecture (*InputStream*) et un pour l'écriture (*OutputStream*). Pour interagir avec la socket il faut donc lire depuis, ou écrire dans, un de ces streams.

1. En utilisant un *BufferedReader* et un *InputStreamReader*, modifiez votre serveur pour qu'il lise une unique ligne depuis la socket et l'affiche à l'écran ;

2. Si aucune ligne n'est disponible, que se passe-t-il ?

3. Testez à nouveau votre serveur en simulant un client via la commande *telnet* ;

4. À l'aide de *PrintWriter* modifiez votre client pour qu'il écrive un message dans la socket ;

5. Testez votre client.

5 Bouclons jusqu'à la fin

Notre serveur ne peut traiter qu'un seul et unique message, ensuite il s'arrête. Nous allons maintenant le modifier afin de lire autant de messages que souhaité.

1. Modifiez la partie du serveur qui s'occupe de lire les messages afin qu'il ne s'arrête que lors d'un problème (lecture *null*) ;

2. Modifiez votre client pour qu'il envoie plusieurs messages lorsqu'il est connecté au serveur ;

3. Testez à nouveau votre client et serveur ;

4. Modifiez le serveur pour qu'il s'arrête si il lit la chaîne de caractères « *stop* ».

6 Communications bi-directionnelles

Jusqu'à maintenant le serveur est le seul à pouvoir recevoir et afficher un message. Cependant les communications peuvent être bi-directionnelles. C'est ce que nous allons voir ci-après :

1. Modifiez le serveur pour envoyer immédiatement au client la phrase « *Serveur accuse réception de :* » suivi du texte lu lorsqu'un message est réceptionné par le serveur ;

2. Modifiez le client pour qu'il affiche ce qui est lu depuis la socket ;

3. Testez le bon fonctionnement de votre client et serveur.