TP2 : sockets

En Java, un objet d'une des classes *Socket*, *ServerSocket*, *DatagramSocket* ou *MulticastSocket* est appelé socket (dans le langage courant, le mot anglais *socket* peut se traduire par prise). En fait, de manière générale, on appelle sockets des interfaces de programmation réseau qui sont présentes dans de nombreux langage de programmation. Les sockets permettent la communication entre les machines d'un réseau et ne préjugent en général pas du protocole de communication utilisé (i.e elles permettent simplement d'envoyer et de recevoir des messages). Dans le cas particulier de Java les sockets sont une interface de plus haut niveau que dans beaucoup de langages, et en particulier les classes *Socket* et *ServerSocket* (qui sont le sujet de ce TP) utilisent le protocole TCP : elles permettent donc la communication point à point et on peut considérer que les messages ne se perdent pas et arrivent dans l'ordre (les autres classes utilisent le protocole UDP).

1 Objectif du TP

L'objectif de ce TP sera d'implanter un chat rudimentaire, fonctionnant selon un protocole très simple. Ce chat fonctionnera selon le modèle client/serveur : un serveur se chargera de recevoir tous les messages des utilisateurs et de les afficher, chaque utilisateur disposera d'un client lui permettant d'envoyer des messages au serveur.

Dans un premier temps, vous utiliserez la classe *Socket* pour programmer un client, à l'aide duquel vous vous connecterez à un serveur qui sera mis en place sur la machine enseignant. Ensuite vous utiliserez la classe *ServerSocket* pour programmer vous même un serveur.

Le protocole que nous utiliserons pour la connexion d'un client au serveur sera le suivant :

1. Lorsque le client entre en contact avec le serveur, celui-ci lui envoie le message `tki?`,
2. Le client doit répondre en proposant un nom d'utilisateur (par exemple pour proposer le nom *Alice*, le client enverra le message `Alice`),
3. Si ce nom est déjà pris, le serveur répond par le message `tkn!tki?` et on retourne à l'étape 2,
4. Si ce nom n'est pas pris, le serveur répond par le message `ok`,
5. Le client peut alors envoyer librement des messages au serveur, qui seront affichés par celui-ci.

2 Client : la classe Socket

La classe *Socket* fait partie du paquet **java.net** (on importera aussi le paquet **java.io** pour la communication entre client et serveur). Elle permet de créer des objets qui vont démarrer une communication avec un serveur, en indiquant l'adresse de ce serveur sur le réseau ainsi que le port qu'il utilise pour communiquer.

On créé un objet socket ouvrant une connexion au serveur dont l'adresse est *adresseServeur* sur le port *numeroPort* de la façon suivante :

```
try {
  Socket sock = new Socket(adresseServeur, numeroPort);
} catch (UnknownHostException e) {
  //on ne trouve pas le serveur
} catch (IOException e) {
  //la connexion echoue
}
```

On peut alors dialoguer avec le serveur via un flux d'entrée (sur lequel on reçoit les messages du serveur) et un flux de sortie (sur lequel on envoie des messages au serveur). Ces flux sont accessibles par les appels de méthodes suivants :

```
InputStream fluxEntree = sock.getInputStream(); //flux d'entree
OutputStream fluxSortie = sock.getOutputStream(); //flux de sortie
```

On peut alors écrire sur le flux de sortie, octet par octet, avec la méthode *write* et lire sur le flux d'entrée, toujours octet par octet, avec la méthode *read*. Pour se simplifier la vie, et pouvoir lire des chaînes de caractères directement, on peut procéder de la manière suivante :

```
BufferedReader entree = new BufferedReader(new InputStreamReader(fluxEntree))
PrintWriter sortie = new PrintWriter(fluxSortie, true);
String entreeLue = entree.readLine(); //lecture du flux entrant
sortie.println("Salut␣!"); //ecriture sur le flux sortant
```

Question. Écrivez un client qui communique avec le serveur mis en place sur la machine enseignant en utilisant le protocole décrit plus haut.

3 Serveur : la classe ServerSocket

Un serveur doit écouter sur un port jusqu'à recevoir une demande de connexion d'un client. Ceci se fait à l'aide d'un objet de la classe *ServerSocket* :

```
try {
   ServerSocket serverSock = new ServerSocket(numeroPort);
} catch (IOException e) {
  //a ne pas oublier
}
```

La méthode *accept* permet ensuite de recevoir les demandes de connexion des clients. Elle crée un objet de la classe *Socket* à l'aide du quel le serveur pourra communiquer avec le client (exactement de la même façon que le client communiquait avec le serveur dans la partie précédente).

```
Socket sock = serverSock.accept();
InputStream fluxEntree = sock.getInputStream();
OutputStream fluxSortie = sock.getOutputStream();
```

Question. Écrivez un serveur capable de communiquer avec votre client selon le protocole décrit plus haut.

Question. Modifiez votre serveur pour qu'il puisse communiquer en même temps avec plusieurs clients. Pour cela, chaque nouveau client connecté sera engendrera la création d'un thread pour gérer sa communication avec le serveur.

4 S'il vous reste du temps

Question. Améliorez votre serveur de façon à ce qu'il retransmette à tous les clients les messages envoyés par les autres clients.

Question. Améliorez votre client pour qu'il affiche les messages retransmis par le serveur.