Fiche n°1

Sockets en mode connecté

Cet article présente comment créer un client et un serveur en utilisant une socket en mode connecté, et envoyer et lire une chaîne de caractères.

1 Le programme serveur

Dans un premier temps, le serveur doit créer une socket de connexion qui permettra d'accepter des connexions de la part des clients. Le port d'écoute est spécifié ici dans un attribut `portEcoute`.

```
ServerSocket socketServeur = null;
try {
  socketServeur = new ServerSocket(portEcoute);
} catch(IOException e) {
  System.err.println("Création de la socket impossible : " + e);
  System.exit(0);
}
```

Une fois la socket créée, le serveur se met en attente d'une connexion à l'aide de la méthode `accept`.

```
Socket socketClient = null;
try {
  System.out.println("En attente d'une connexion d'un client...");
  socketClient = socketServeur.accept();
} catch(IOException e) {
  System.err.println("Erreur lors de l'attente d'une connexion :" + e);
  System.exit(0);
}
```


Pour accepter plusieurs clients, il faut que le code précédent puisse être exécuté plusieurs fois, en mettant en place une boucle, par exemple.

Pour lire ou écrire des données dans la socket de communication, nous choisissons d'encapsuler les flux d'entrée/sortie dans des flux de type `InputStreamReader` et `OutputStreamReader`, eux-mêmes encapsulés dans des flux de type `BufferedReader` et `PrintWriter`. Nous pourrons ainsi lire facilement des données de type primitif (entier, réels) ou des chaînes de caractères.

```
BufferedReader input = null;
PrintWriter output = null;
try {
  input = new BufferedReader(new InputStreamReader(socketClient.
     getInputStream()));
  output = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
     socketClient.getOutputStream())), true);
} catch(IOException e) {
  System.err.println("Association␣des␣flux␣impossible␣:␣" + e);
  System.exit(0);
}
```

Le serveur peut alors recevoir des données (ici une chaîne de caractères) :

```
String message = "";
try {
  message = input.readLine();
} catch(IOException e) {
  System.err.println("Erreur␣lors␣de␣la␣lecture␣:␣" + e);
  System.exit(0);
}
System.out.println("Lu:␣" + message);
```

Ou bien envoyer des données (ici, une chaîne de caractères) :

```
String message = "Bonjour";
System.out.println("Envoi:␣" + message);
output.println(message);
```

2 Le programme client

Le client doit créer une socket de communication à l'aide de la classe `Socket`. Le constructeur prend en paramètre l'adresse (ou un nom de domaine) et un port d'écoute (ici `portEcoute`, un attribut de la classe).

```
Socket socket = null;
try {
  socket = new Socket("localhost", portEcoute);
} catch(UnknownHostException e) {
  System.err.println("Erreur␣sur␣l'hôte␣:␣" + e);
  System.exit(0);
} catch(IOException e) {
  System.err.println("Création␣de␣la␣socket␣impossible␣:␣" + e);
  System.exit(0);
}
```

Comme pour le serveur, nous encapsulons les flux d'entrée/sortie de la socket dans des flux qui nous permettront de lire des données de type primitif ou des chaînes de caractères.

```
BufferedReader input = null;
PrintWriter output = null;
try {
  input = new BufferedReader(new InputStreamReader(socket.getInputStream
     ()));
  output = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
     socket.getOutputStream())), true);
} catch(IOException e) {
  System.err.println("Association des flux impossible : " + e);
  System.exit(0);
}
```

Comme pour le serveur, le client peut alors envoyer ou recevoir des données.

L'envoi et la réception doivent respecter un dialogue d'échange entre le client et le serveur. La lecture d'un côté doit correspondre à une écriture de l'autre.

3 Exécution

Pour tester le client et le serveur, dans un premier temps, vous devez compiler les classes puis exécuter le serveur. Le client ne peut être démarré qu'une fois le serveur démarré. À noter que le serveur s'arrête une fois le message lu.

Le numéro de port d'écoute du serveur est spécifié via une constante. S'il est déjà utilisé, il est possible de le modifier (dans les deux classes). Une première amélioration consiste à spécifier le numéro de port en argument du serveur (ce qui évite de recompiler la classe à chaque changement). Une autre solution consiste à laisser le système choisir le numéro port automatiquement (comme pour le client) et de l'afficher à l'écran pour pouvoir le spécifier ensuite en argument au client.