TD nº 5 - Client-serveur

Le but de ce TD est de réaliser une application client-serveur simple. Dans cette application, il y aura deux exécutables. L'un est le serveur qui attend des requêtes de la part des clients et pourra dialoguer simultanément avec plusieurs clients. L'autre est le client qui envoie ses requêtes au serveur.Pour réaliser l'application, nous suivons un dévéloppement progressif. On commence par écrire une application dans laquelle le serveur communique avec un unique client situé sur la même machine. Enfin, on modifie le serveur pour qu'il soit capable de communiquer avec plusieurs clients à la fois, se trouvant sur des machines différentes.

Exercice 1. *Le cas simple*

Nous allons nous limiter dans un premier temps à un seul client à la fois. Le serveur a une seule tâche (la fonction main par exemple) qui crée un `ServerSocket` pour écouter sur le port de l'application (on utilisera le port 6020 dans ce TD). Lorsqu'un client se connecte, le serveur ouvre un `Socket` pour communiquer.

```
import java.io.*;
import java.net.*;
public class Serveur {
   public static void main(String[] args) throws Exception {
        ServerSocket s = new ServerSocket(6020);
           System.out.println("START");
        Socket soc = s.accept();
        BufferedReader ins = new BufferedReader(
                             new InputStreamReader(soc.getInputStream()) );
        PrintWriter outs = new PrintWriter( new BufferedWriter(
                            new OutputStreamWriter(soc.getOutputStream())), true);

       // insertion de la boucle du serveur ici

        ins.close();
        outs.close();
        soc.close();
   }
}
```

1. Lisez attentivement le code ci-dessus correspondant à la classe `Serveur` et vérifiez que vous comprenez la signification de chaque instruction.

2. Complétez la classe `Serveur` pour que le serveur exécute une boucle infinie dans laquelle il lit les messages envoyés par le client et répond en renvoyant le même message préfixé de la chaine " Recu : ".

De son côté, le client exécute aussi une seule tâche. Pour se connecter au serveur, il doit connaître l'adresse de la machine et le port d'écoute. L'adresse sera passée en argument lors de l'exécution (donc à récupérer dans le tableau d'arguments de la fonction main), le port sera toujours 6020.

3. En vous inspirant de la classe `Serveur`, écrivez la classe `Client` qui se connecte sur le port 6020 du serveur dont l'adresse est donnée en argument, puis envoie successivement 10 messages " message 1 ", " message 2 ", etc. et affiche après chaque message envoyé la réponse du serveur.

4. Testez votre programme en lançant successivement un serveur et un client. N'oubliez pas de donner l'adresse du serveur lors de l'exécution du client. Vous pouvez utiliser l'adresse " localhost " si le client et le serveur sont lancés sur la même machine.

5. Modifiez le serveur pour qu'il interrompe sa boucle infinie lorsqu'il reçoit le message " stop ". Faites en sorte que le client envoie le message " stop " après avoir envoyé ses 10 premiers messages et vérifiez que votre application se termine correctement.

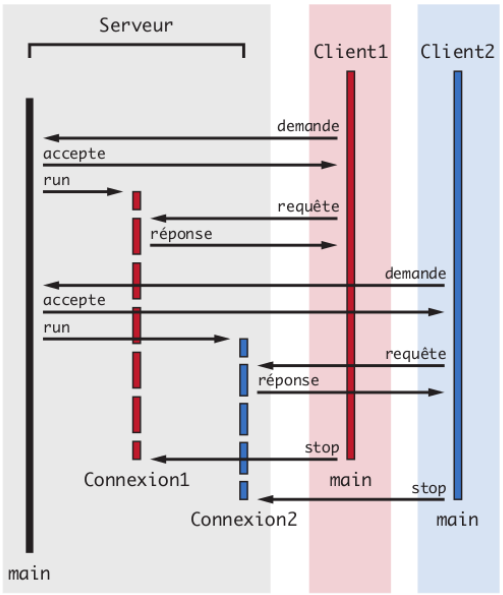
6. Modifiez le client pour qu'il demande à l'utilisateur d'entrer les messages à envoyer au serveur depuis le console et qu'il affiche les réponses du serveur dans la même fenêtre. Le nombre de messages du client n'est plus fixé à l'avance. Le client envoie des messages jusqu'à ce que l'utilisateur entre le message " stop " ce qui doit alors provoquer la fermeture du client après avoir envoyé le message au serveur.

Exercice 2. *Serveur multi-tâche*

Dans cet exercice on va réorganiser le serveur pour qu'il puisse communiquer avec plusieurs clients simultanément.

1. En l'état, que se passe-t-il si un client essaie de se connecter au serveur alors que le serveur a déjà ouvert une connexion avec un autre client (vous pouvez faire le test) ?

Pour gérer plusieurs connexions simultanément, le serveur va exécuter une tâche différente pour chaque client qui se connecte. On ajoute donc une classe `Connexion` qui implémente `Runnable` dont la méthode `run()` est responsable de la communication avec un client donné. La méthode `main()` du serveur se contente donc maintenant de créer un `ExecutorService` puis d'effectuer une boucle infinie dans laquelle le serveur écoute sur le port 6020 et, quand un client s'y connecte, ouvre une socket et exécute une nouvelle `Connexion` qui va communiquer avec le client sur la socket nouvellement crée. La figure illustre ce fonctionnement.



2. Ecrivez la classe `Connexion` implémentant l'interface `Runnable` dont le constructeur prend en argument une `Socket` (qui aura déjà été ouverte et connectée au client) et dont la méthode `run()` se charge de la communication avec le client.

3. Modifiez la classe `Serveur` pour que la méthode `main()` se contente maintenant d'écouter sur le port 6020 et exécute une nouvelle `Connexion` lorsqu'un client se connecte. Une autre variante possible : l'activité principale du serveur peut aussi être un tâche gérée comme les autres...

Remarque : Idéalement, les instructions de la méthode `main()` devraient être placées dans un bloc `try` suivi d'une clause `finally` pour éviter le blocage du port 6020 lors de fermetures intempestives de l'application. Le bloc `finally` doit fermer la socket d'écoute du serveur.

4. Testez votre application en connectant plusieurs clients à un même serveur.

5. Vérifiez que votre application fonctionne bien à travers le réseau en connectant des clients exécutés sur des machines différentes (si vous et votre voisin avez bien suivi les instructions du sujet, vos clients devraient pouvoir se connecter sur leur serveur, et inversement).

Exercice 3. *Chatrooms*

Dans ce dernier exercice, on va utiliser l'architecture client-serveur développé dans ce TD pour écrire une application de chat permettant à plusieurs utilisateurs de participer à une même discussion. On veut que les clients puissent se connecter au serveur et envoyer des messages. Le serveur enverra alors le message à tous les utilisateurs connectés (broadcast). Pour que la conversation soit compréhensible, il faut que chaque client soit identifié par un nom et que le nom d'un client apparaisse devant chacun de ses messages.

1. Modifiez votre application pour réaliser un programme de chat.

Indication : Il faut gérer un tableau contenant l'ensemble des `Connexions` ouvertes à tout instant (mis à jour par le serveur à chaque nouvelle connexion) et faire en sorte que tout message reçu soit transmis à toutes les `Connexions`.

2. (bonus) Ajoutez des fonctionnalités supplémentaires à votre application : possibilités de gérer plusieurs conversations séparées en demandant aux clients de se connecter à un "salon" à leur arrivée, possibilité d'envoyer des messages privés à un utilisateur donné, etc.