Tp 6 / Programmation réseau.

**Exercice 1** **-** [**Un client Echo en TCP**](http://igm.univ-mlv.fr/~duris/RESEAU/td6-mi03.html)

Écrire un client Echo s'appuyant sur TCP. Il faut pour cela établir une connexion avec un serveur donné, sur son port TCP , en créant un objet de la classe Socket qui établit la connexion. On peut alors lire ou écrire sur les flots d'entrée ou de sortie associés à cet objet. La connexion se termine en fermant la socket par la méthode close().

**Exercice 2** **-** [**Serveur de mise en majuscules**](http://igm.univ-mlv.fr/~duris/RESEAU/td6-mi03.html)

Écrire un programme, UpperCaseTCPServer, représentant un serveur TCP itératif renvoyant les chaînes de caractères envoyées par des clients après les avoir mis en majuscule.

Ce serveur crée un socket serveur (objet de la classe java.net.ServerSocket), puis est *"démarré"*. Il attend alors une connexion (dite *pendante*) d'un client, *via* la méthode accept() appelée sur l'objet ServerSocket. Lorsque des clients contactent le serveur, l'une des connexions correspondantes est *élue* par la méthode accept(), qui retourne un objet de la classe Socket. Celle-ci est alors dite *socket de service*. Le serveur peut ainsi satisfaire la ou les requêtes successives émises par le client sur le socket de service. Puisque notre serveur est *itératif*, lorsque les différentes requêtes de ce client sont traitées, la socket de service peut être fermée, et une nouvelle connexion pendante peut être élue comme socket de service par un appel à accept() sur l'objet ServerSocket.

Le principe du serveur UpperCaseTCPServer, pour le traitement des requêtes d'un socket de service qu'il vient d'accepter, est le suivant:

1. Envoyer au client un message d'accueil, précisant les modalités d'envoi des requêtes. Par exemple, les chaînes doivent être envoyées ligne par ligne, et l'envoi d'une ligne contenant uniquement un point (".") termine la session.
2. Pour chaque ligne reçue par le client, le serveur la met en majuscule et la renvoie au client.
3. Lorsqu'une ligne ne contenant qu'un point est reçue, le serveur renvoie cette ligne puis ferme le socket de service : il termine ainsi sa session avec ce client.

À son lancement, le serveur affiche son adresse et son numéro de port d'attachement local, afin que les clients puissent accéder à son service.

**Exercice 3** **-** [**Un client pour le service de mise en majuscule**](http://igm.univ-mlv.fr/~duris/RESEAU/UpperCaseTCPClient.java)

Écrire une classe UpperCaseTCPClient représentant un client du service UpperCaseTCPServer. L'exécution de ce programme acceptera en argument sur la ligne de commande le nom du serveur et le numéro de port ou le service est accessible. Une fois connecté au serveur (à la réception du message d'accueil de celui-ci), le programme client permettra d'envoyer des chaînes de caractères au format "UTF-16LE" (saisies au clavier) vers le serveur et d'afficher les réponses correspondantes.

**Exercice 4** **-** [**Proxy**](http://igm.univ-mlv.fr/~duris/RESEAU/td6-mi03.html)

Écrire une application Proxy qui permet de relayer entre un client et un serveur toutes les données transitant sur une connexion TCP. Le proxy est lancé sur une machine ProxyMachine en fournissant un numéro de port local (ProxyPort) ainsi que l'adresse de la socket distante du serveur auquel il doit relayer les données (RemoteMachine:RemotePort).

Lorsque ProxyMachine recoit sur son port ProxyPort une demande de connexion depuis un client, elle accepte cette connexion et doit à son tour demander l'établissement d'une connexion entre elle-même et le port RemotePort de RemoteMachine. Une fois ces deux connexions établies, le proxy crée et démarre deux processus légers chargés de relayer les informations circulant, d'une part, entre le client et la machine distante et, d'autre part, entre la machine distante et le client.

**Exercice 5** **-** [**Serveur UpperCase concurrent**](http://igm.univ-mlv.fr/~duris/RESEAU/.java)

Reprendre les spécifications du serveur UpperCaseTCPServer et écrire un serveur UpperCaseTCPConcurentServer qui délègue le traitement des requêtes correspondant aux sessions établies avec chaque client à des processus légers distincts. On pourra par exemple créer 10 processus légers, chacun chargé de répondre à une socket de service élue. Lorsque les 10 processus légers sont occupés, les clients ne peuvent plus être servis tant que l'une au moins des sessions établies n'est pas terminée.

Exercice 6 : Client/Serveur

1. Écrire un client et un serveur en Java. Le serveur choisira un entier aléatoirement entre

1 et 100. Et répondra « + » si son entier est plus grand que celui envoyé par le client,

« - » s’il est plus petit et « = » s’il est égal avant de fermer la connexion. Le client

affichera ce qu’il reçoit du serveur.

2. Sans changer le serveur, modifier votre client pour que ce soit plus lisible pour l’utilisateur.

3. Tester votre client avec le serveur d’un de vos camarades, et inversement.

Indications : La classe InetAddress vous permet d’avoir des informations sur la machine

où vous exécutez votre programme, par exemple :

InetAddress.getLocalHost().getHostName();.

Pour créer le serveur utilisez la classe ServerSocket.