TCP, Socket, SocketChannel

Exercice 1 - Un client Echo en TCP

Ecrire un client Echo s'appuyant sur TCP. Il faut pour cela établir une connexion avec un serveur donnée, sur son port TCP 7, en créant un objet de la classe `java.net.Socket`.
On peut alors lire ou écrire sur les flots d'entrée ou de sortie associées à cet objet. (`getInputStream()/getOutputStream()`).
La connexion se termine en fermant la socket par la méthode `close()`.

Exercice 2 - Client Echo avec des channels

Ecrire une autre version du client TCP en utilisant la classe `SocketChannel`
La lecture ou l'écriture s'effectue alors en utilisant un `ByteBuffer`.

Exercice 3 - Serveur de mise en majuscules

Ecrire un programme, `UpperCaseTCPServer`, implantant un serveur TCP itératif renvoyant les chaînes de caractères envoyées au format `"ISO8859 1"` par des clients après les avoir mis en majuscule.
Ce serveur crée une socket serveur (objet de la classe `java.nio.channels.ServerSocketChannel`), puis est "démarrée". Il attend alors une connexion d'un client, via la méthode `accept()` appelée sur l'objet `ServerSocketChannel`.
Lorsque des clients contactent le serveur, l'une des connexions correspondantes est élue par la méthode `accept()`, qui retourne un objet de la classe `SocketChannel`. Celle-ci est alors dite socket de service. Le serveur peut ainsi satisfaire la ou les requêtes successives émises par le client sur la 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 `ServerSocketChannel`.
Le principe du serveur `UpperCaseTCPServer`, pour le traitement des requêtes d'une socket de service qu'il vient d'accepter, est le suivant :

1. Envoyer au client un message de bienvenue.
2. Les données doivent être envoyées ligne par ligne, et une ligne contenant uniquement '.' termine la session. Pour chaque ligne reçue par le client, le serveur la met en majuscule et la renvoie au client.
3. Une ligne contenant uniquement '.' entraîne la fermeture la socket.
4. Si la connexion reste trop longtemps ouverte sans activité, elle est fermée par le serveur en utilisant un `setSoTimeout()` sur la socket de service.
   A 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 4 - Proxy

Ecrire 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` reçoit 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-m^eme 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

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 que vous gérerez vous-même. On pourra par exemple créer 3 processus légers, chacun chargé de répondre à une socket de service élue.
Lorsque les 3 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 - Serveur UpperCase avec un thread pool

Ecrire une autre version du serveur `UpperCaseTCPConcurrentServer` qui utilise un pool de processus légers créé avec la méthode statique `Executors.newFixedThreadPool()`.