PR6 – Programmation réseaux
TP n° 3 : Protocoles : découverte (en Java)

Dans ce TP, toutes les transmissions seront faites en mode connecté TCP.

En Java, il est possible d'ouvrir une connexion à un service de *port* donné sur une *machine* donnée en utilisant la classe `java.net.Socket`. Par exemple :

```
Socket s = new Socket(machine,port);
```

Pour créer les serveurs, utilisez la classe `ServerSocket` en écoutant sur *localhost*.
Les méthodes `getOutputStream()` et `getInputStream()` permettent de retrouver des flux d'entrées/sorties (respectivement de type `OutputStream` et `InputStream`) pour écrire et lire sur cet objet de communication. De façon à manipuler ces objets, regardez comment les utiliser avec les classes `PrintStream`, `InputStreamReader` et `BufferedReader`.

Les méthodes sont décrites dans le cours 2, à partir du transparent 9. Assurez-vous de bien avoir compris comment cela fonctionne avant de vous lancer dans du code !

Exercice 1 : Ports ouverts

Écrire un programme `Ports` en java qui teste quels ports, dont les numéros sont dans une plage donnée, sont ouverts sur une machine donnée. Par exemple la commande :
`$ java Ports 0 1024 lucien`
affiche sur la sortie standard tous les numéros de ports entre 0 et 1024 ouverts sur lucien et sur la sortie erreur ceux qui ne sont pas ouverts.

Exercice 2 : Client daytime

Écrire un programme `Jdaytime` en java qui se connecte au service `daytime` d'une machine donnée en paramètre et récupère la date pour l'afficher à l'écran.

Exercice 3 : Serveur de mise en majuscules

Écrire un programme, `ServeurMajuscule`, représentant un serveur TCP itératif renvoyant les chaînes de caractères envoyées par des clients après les avoir mis en majuscule.
Le principe du serveur pour le traitement des requêtes d'une socket de service qu'il vient d'accepter, est le suivant :

- 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.
- Pour chaque ligne reçue par le client, le serveur la met en majuscule et la renvoie au client.
- Lorsqu'une ligne ne contenant qu'un point est reçue, le serveur ferme la socket de service afin de terminer 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.
Tester le serveur à l'aide de `telnet`.

Exercice 4 : Client «de mise en majuscules»

Écrire un programme, `ClientMajuscule` permettant d'utiliser le service du `ServeurMajuscule` sur les chaînes de caractères passées en argument de la commande.

Exercice 5 : Client/Serveur « Plus ou Moins »

Le but de cet exercice est de programmer le jeu bien connu du « Plus ou Moins », version client/serveur. 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. Le client et le serveur communiquent en envoyant des chaînes de caractères comme dans les exercices précédents (*i.e.* à l'aide de `println`). Dans un premier temps on se contentera du client `telnet` et on écrira le serveur.

1. Écrire le serveur en Java. Dans cette version, le serveur est en attente de clients, et lorsqu'un client se présente il :
   - choisit un nombre aléatoire puis répète jusqu'à ce que le client ait trouvé la solution
     - attend la proposition du client
     - renvoie le symbole correspondant (`+ | - | =`)
   - ferme la connexion
   - attend un nouveau client
2. Tester votre serveur avec `telnet`.
3. Écrire le client en Java. Une fois la connexion établie, le client demande une entrée utilisateur et l'envoie au serveur. Il affiche la réponse du serveur telle quelle et recommence jusqu'à ce que la solution ait été trouvée.
4. Tester le client (sans changer le serveur !).
5. Modifier votre client pour que ce soit plus lisible pour l'utilisateur (texte d'invite de saisie, réponses en langue française développée...)
6. Tester votre client avec le serveur d'un de vos camarades, et inversement.
7. Vérifier la solidité du serveur de sorte qu'il continue à fonctionner malgré les aléas (envoi d'autre chose qu'un nombre, interruption brutale du client).