2ème année DUT Informatique
IUT Aix-en-Provence

TP3 Programmation Répartie
Sockets en Java

Safa Yahi

Exercice 1 (Client TCP daytime)

Ecrivez une classe **ClientTcpDaytime** qui implémente, comme son nom l'indique, un client TCP pour le service daytime.
Cette classe admet **deux attributs** : une chaine de caractères (hostname) qui représente l'adresse IP (sous format textuel) ou le nom de machine du serveur et un nombre entier (port) qui désigne le port sur lequel écoute le serveur.
Pour initialiser ces attributs, on définit **un constructeur** ayant deux paramètres.
De plus, cette classe admet une méthode **public void lancerBR()** où le client se connecte au serveur et récupère la date et l'heure en utilisant la classe BufferedReader.

=> Testez la classe ClientTcpDaytime avec le serveur daytime TCP en C du TP1.

Exercice 2 (Client TCP echo)

Dans cet exercice, nous nous intéressons au service "echo". Comme "daytime", il s'agit d'un service assez simple utilisé souvent pour illustrer l'implémentation de sockets. Par défaut, le port associé est le port 7.

Ecrivez une classe **ClientTcpEcho** qui implémente un client TCP en java pour le service **echo**
Plus précisément :

- Au niveau du client, l'utilisateur saisit une chaine de caractères (lecture depuis le clavier).
- Le client envoie ensuite cette chaine au serveur.
- Le serveur envoie une nouvelle chaine au client. En général, c'est la même chaine, mais ici on veut qu'elle soit transformée en majuscule.
- le client reccupère la nouvelle chaîne envoyée par le serveur et il l'affiche.

Ce processus est réitéré jusqu'à ce que l'utilisateur entre la chaine "quit" ou que l'un des deux se déconnecte.

Cette classe admet **deux attributs** : une chaine de caractères (hostname) qui représente l'adresse IP (sous format textuel) ou le nom de machine du serveur et un nombre entier (port) qui désigne le port sur lequel écoute le serveur.

Pour initialiser ces attributs, on utilise **un constructeur** de deux paramètres.
Cette classe possède une méthode **public void lancerBW()** où le client utilise la classe BufferedWriter pour envoyer les données.

=> Testez la classe ClientTcpEcho avec le serveur echo indiqué par votre enseignant(e).

Exercice 3 (Serveur TCP echo)

Ecrivez une classe **ServeurTcpEcho** qui implémente un serveur TCP echo à une exception près : la chaine retournée par le serveurdoit être en majuscule. On suppose que ce serveur traite un nombre donné de clients (soit nbClients).
=> Testez cette classe avec un client telnet ensuite avec le client TCP echo que vous avez implémenté précédemment, en local et avec votre voisin.

Exercice 4 (Serveur TCP echo Multithread)

Implémentez une classe ServeurTcpEchoMulti qui représente un serveur TCP Echo et qui permet de traiter plusieurs clients en même temps en utiliant les threads.

Exercice 5 (Client SMTP)

AVERTISSEMENT !!!

Le but de cet exercice est purement pédagogique. Contentez-vous de faire uniquement ce qui est demandé dans l'énoncé. Il vous est interdit de rajouter tout code visant une utilisation malsaine d'un serveur SMTP.

Le but de cet exercice est d'écrire un client SMTP en Java en utilisant les sockets TCP.

Le protocole SMTP (Simple Mail Transfer Protocol) permet le transfert des mails depuis la machine de l'émetteur vers la machine qui héberge la boite email du destinataire. Ce protocole est décrit par la RFC 2821. Il est basé sur un échange bien défini de messages entre le client et le serveur.

Les messages (commandes) du client les plus fréquemment utilisés sont :

- EHLO : pour l'adresse IP ou le nom de la machine sur laquelle s'exécute le client
- MAIL FROM : pour le mail de l'émetteur
- RCPT TO : pour le mail du destinataire
- DATA: pour le sujet, le corp du message, etc. Notons que la partie DATA doit se termnier par un "." afin d'indiquer la fin du messsage.

Côté serveur, les réponses commencent par un code de trois chiffres. Les codes qui débutent par 4 ou 5 correspondent à des messages d'erreurs. Sinon, il s'agit de messages positifs.

Aussi, les réponses du serveur peuvent tenir sur plusieurs lignes. Afin de distinguer la dernière ligne des autres lignes, on utilise un caractère spécial après le code de trois chiffres. Plus précisément, pour toutes les lignes hormis la dernière, on met le caractère "-" (moins) après le code. Pour la dernière ligne, il s'agit du caractère "SP" (un blanc). Voici un exemple d'une réponse multiligne :

```
250-mx0.univ-amu.fr   //  première ligne (caractère "-" après le code)
250-8BITMIME          // deuxième ligne (caractère "-" après le code)
250 SIZE 20971520     // dernière ligne  (pas de caractère "-" après le code)
```

Avant d'écrire votre client, testez d'abord le protocole SMTP en passant par telnet. Envoyez un message de votre choix à l'utilisateur garfield qui a une boite aux mettres sur la machine indiquée par votre enseignant(e). Vous pouvez vous inspirer de l'exemple suivant où on se connecte à la

machine 139.124.187.29 via le port 25 pour envoyer un message à **riri** de la part de **toto**. Les messages envoyés par le client sont en bleu. Les messages en vert sont ceux du serveur.

Exemple :

```
vdebiut:~# telnet 139.124.187.29 25
Trying 139.124.187.29...
Connected to localhost.
Escape character is '^]'.
220 vdebiut ESMTP Postfix (Debian/GNU)
EHLO vdebiut
250-vdebiut
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
MAIL FROM: <toto>
250 2.1.0 Ok
RCPT TO: <riri>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
SUBJECT: Hello
Bonjour Riri
Comment ça va ?
Je dois écrire un client SMTP
A plus
.
250 2.0.0 Ok: queued as 9E32913A51
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

Imprémentez ensuite la classe ClientSmtp qui admet le **constructeur** et la méthode **send** suivants :

- **public ClientSmtp(String serveurSmtp, int port, String hostname)** où :
  - “serveurSmtp” est l'adresse IP ou le nom du serveur SMTP du domaine du destinataire
  - “port” désigne le port sur lequel écoute le serveur SMTP (par défaut, c'est le port 25).
  - “hostname” est l'adresse IP ou le nom de la machine du client SMTP.

- **public void send(String from, String to, String subject, String body)** où :
  - “from” indique l'adresse email de l'émetteur du message
  - “to” indique l'adresse du destinataire
  - “subject” désigne le sujet du message
  - “body” indique le corp du message

Dans la fonction send(), le dialogue entre le client et le serveur se fait à tour de rôle. Autrement, dit, le client n'envoie pas toutes les commande d'un seul coup. Il doit à chaque fois prendre en compte la réponse du serveur. S'il s'agit d'un message d'erreur, alors le client arrête interrompe la connexion.