1 Exercices de compréhension

Voici quelques exercices pour la prise en main de *telnet* (ou *nc*). Donnez les commandes en shell et décrivez le résultat.

1.1 Donnez l'échange de texte (entre navigateur et serveur web) pour consulter la page :

```
http://r305.parlenet.org/debut
```

```
GET /debut undefined
Host: r305.parlenet.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/
*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

2 Validation des outils

Vous pouvez rencontrer des difficultés de connexions avec un système gérant en même temps IPV6 et IPV4. Dans ce cas utilisez l'option :

```
java -Djava.net.preferIPv4Stack=true -cp …
```

2.1 Teste de RecvCmd

Téléchargez le programme *recvcmd.java* sur le site http://r305.parlenet.org} et testez-le avec le port 8080. Comment vous connectez-vous à lui ?

On doit utiliser RecvData.java

```
omergs@omerpc:~/Bureau/R3.05/TP2$ java RecvCmd.java 8080
Start server on port 8080
Client comming from /192.168.104.25:57170
test2
Bonjour
Socket closed
```

Donnez des traces.

Le code serveur.

Le code client.

```
rayanne@rian:~/Documents/BUT2/R305/TDP2/java/src$ java SendCmd.java 192.168.104.205 8080 test2
Bonjour
Socket closed
```

3 Compréhension du logiciel

3.1 Donnez le pseudo-code de la tâche qui attente des connexions des clients.

C'est la boucle for(; ;), qui est une boucle infini, qui attend la connexion des clients.

```
for (;;) {
    Socket call = serverSocket.accept ();
```

3.2 Donnez le pseudo-code qui attend les lignes d'un client.

```
For (;;) {
  String line1 = in.readLine ();
  if (line1 == null)
  break;
```

3.3 Décrire les risques de conflit qui peuvent apparaître.

Si deux clients essaye de se connecter au serveur en même temps seul le client qui se connectera en premier aura une communication avec le serveur rentrera dans une boucle infini.

3.4 Combien faut-il écrire de programmes (ainsi que ce qui peut être réutilisé d'Unix) ?

Il faut écrire deux programmes le premier pour le client, le second pour le serveur. On peut donner le programme client à plusieurs personnes pour faire un échange de message à plusieurs.

4 Réalisation du logiciel

Consultez l'API Java concernant les classes :

```
String, Integer, Vector<E>, ServerSocket, Socket, IOException,
Thread, PrintStream, BufferedReader, InputStreamReader
```

Pour réaliser cet exercice, j'ai utilisé les mots clefs Java :

5 **break, catch, class, continue, extends, for, if, import, int, new, package, private, public, static, synchronized, this, true, try, void**

Réalisez le programme par étape en insérant votre code en dessous des questions suivantes.

4.1 Faire un lanceur qui prenne en argument le port d'écoute.

```
public Server(int port) throws IOException {
        serverSocket = new ServerSocket(port);
        System.err.println("Server started on port " + port);
}
```

4.2 Écrire le constructeur du serveur et l'attente de nouveaux utilisateurs.

```
public void acceptConnections() {
    try {
      while (true) {
        Socket client = serverSocket.accept();
        User newUser = new User(client);
        connectedUsers.add(newUser);
        System.err.println("User connected: " + newUser);
        new Thread(new UserHandler(newUser, this)).start();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

4.3 Écrire la gestion de l'ensemble des utilisateurs dans le serveur. Comment a-t-on réglé le problème d'accès concurrence à la base des utilisateurs ?

```
public void run() {
    try {
      String message;
      while ((message = in.readLine()) != null) {
        System.out.println("Received from " + user + ": " + message);
        out.println("Server echo: " + message);
      }
    } catch (IOException e) {
      System.err.println("Error with user " + user);
    } finally {
      server.removeUser(user);
      try {
        user.getSocket().close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
```

4.4 Écrire la classe « User » qui représente un utilisateur chez le serveur (attribut et constructeur).

```
private User user;
  private Server server;
  private BufferedReader in;
  private PrintStream out;

  public UserHandler(User user, Server server) throws IOException {
    this.user = user;
    this.server = server;
    Socket socket = user.getSocket();
    this.in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    this.out = new PrintStream(socket.getOutputStream(), true);
  }
```

4.5 Écrire l'activité d'attente de texte envoyé par un utilisateur (écrire la méthode) ?

```
canner = new Scanner(System.in);


        input = scanner.nextLine();
        InetAddress address = InetAddress.getByName (args[0]);
        int port = Integer.parseInt (args [1]);
        byte[] buf = input.getBytes ();
        DatagramPacket packet =
        new DatagramPacket (buf, buf.length, address, port);

        DatagramSocket socket = new DatagramSocket ();
        socket.send(packet);
```

4.6 Donnez des exemples de fonctionnement.

```
omergs@omerpc:~/Bureau/R3.05/TP2$ java Client.java 192.168.104.25 8081
BOnjour
192.168.104.25:33636 send : 192.168.104.205:52404 send : BOnjour
```

4.7 Quelles sont les limites ? Quelles sont les améliorations à apporter ?

Les utilisateurs ne peuvent pas envoyer des messages longues, des images et on ne peut pas differencier les utilisateurs par des pseudo ou un nom. Car la une addresse IP comme identifiant n'est pas interresente.