TP 2
LA COMMUNICATION PAR SOCKETS

Consigne :
- La réalisation des exercices se fera avec un éditeur de texte (ni éclipse, ni netbeans, etc)
- La compilation (javac nomfichier.java) et l'exécution (java nomfichier) en ligne de commande
- La consultation des classes Java à l'adresse *http://download.oracle.com/javase/1.4.2/docs/api/...*

EXERCICE 2.1 : Serveur itératif en mode connecté

```
public class ClientTCP{
      public static void main (String[] args){
            int port=…;
            InetAddress hote=null;
            Socket sc=null;
            BufferedReader in;
            PrintWriter out;

            try{
                  if (args.length>=2){
                                          hote=…
                                          port=…
                  }
                  else{
                        hote = ….
                  }
            }
            catch(UnknownHostException e){
                  System.err.println("Machine inconnue :" +e);
            }
            try{

                  sc = …
                  in = …
                  out = …
                   …
            }
            catch(IOException e){
            System.err.println("Impossible de creer la socket du client : " +e);
            }
            finally{
                  try{
                        sc.close();
                  }
                  catch (IOException e){}
            }
      }

}
```

```
public class ServeurTCP{
      public static void main(String[] args){
            int port=...;
            ServerSocket se;
            Socket ssv=null;
            PrintWriter out;
            BufferedReader in;

            try{
                  se = …
                  ssv = …
                  in = …
                  out = …
                  …
            }
            catch (IOException e){
                  System.err.println("Erreur : " +e);
            }
            finally{
                  try{
                        ssv.close();
                  }
                  catch (IOException e){}
            }
      }
}
```

Vous devez compléter les deux programmes, client et serveur simples, qui vont servir de base de travail. Dans cet exercice, le serveur ne traite qu'un client à la fois.

Dans un premier temps, le serveur et le client n'échangent qu'un message chacun comme décrit dans le scénario suivant :

- Le programme serveur ouvre un socket d'écoute et attend une connexion. Dès qu'un client se connecte, il envoie au serveur une phrase (exemple Bonjour). Le serveur envoie sa réponse (exemple Bienvenu), ensuite il ferme la connexion et termine.

- Le programme client reçoit en paramètre l'adresse IP (de type *InetAddress*) et le port (de type *int*) du serveur. Si rien n'est spécifié, l'adresse IP est *localhost* et le port *4020*. Le client crée un socket TCP pour se connecter sur la machine/port du serveur, ensuite il affiche à l'écran le texte qu'il envoie au serveur, et la réponse du serveur.

1.1 Ecrire le code serveur (qui répond à un seul client et à un seul message) et le code client.

1.2 Modifier les codes précédents afin que le serveur puisse répondre à plusieurs messages d'un client (le client reste connecté au serveur pour 10 échanges d'affilé).

Exercice 2.2 : Serveur concurrent en mode connecté

Modifier le code serveur précédent afin de créer une classe serveur multithread, nommée *ServeurMultiThread*, qui pourra gérer simultanément un nombre illimité de clients. Chaque client possède son propre socket de connexion et tourne dans un thread dédié.

La modification à apporter dans le programme du serveur est une boucle qui crée un nouveau thread de la classe *ThreadClient* à chaque nouvelle connexion et lui passe le socket de connexion. Créer cette classe de thread dédiée à chaque client.

Le code de la classe *ThreadClient* consistera à:

1. envoyer un message d'accueil,
2. attendre une phrase de réponse du client,
3. faire une pause d'une seconde,
4. envoyer une phrase au client,
5. boucler en revenant à l'étape 2
6. sortir de la boucle si la réponse du client est vide ou par exemple la chaîne « Bye »
7. fermer proprement socket et streams

Modifier le code client précédent afin de créer une classe, nommée *ClientMultiThread*, pour que les clients restent connectés au serveur pour 10 échanges d'affilé, en affichant à l'écran les phrases du serveur et en répondant à chaque fois. Le programme principal de chaque client consistera à :

1. recevoir le message du serveur,
2. incrémenter une variable *compteur* qui représente le nombre d'échanges,
3. répondre par : "Je suis le client "+hote+" et j'ai fait "+compteur+" appels" (*InetAddress hote*)
4. faire une pause de 2 secondes,
5. boucler à l'étape 1 dix fois,
6. Terminer par un envoie d'une phrase « vide » ou « Bye »
7. fermer proprement socket et streams

Tester le serveur en lançant plusieurs clients simultanément (au moins 2).

Exercice 2.3 : Internet Relay Chat (IRC)

Le principe d'un IRC est qu'un ensemble de clients, tous connectés au même serveur, puisse dialoguer. Chaque client voit tout ce que disent les autres clients, et peut lui-même parler. Pour distinguer les clients les uns des autres, chacun porte un pseudonyme, qui est repris en entête de chaque message (exemple : toto > comment va tu ?).

Chaque client doit fournir au serveur, à la connexion, un pseudonyme. Le programme client, nommé *ClientIRC*, commence par l'envoi au serveur d'un pseudonyme, qu'il a reçu en paramètre de la ligne de commande.

Le rôle du serveur IRC, nommé *ServeurIRC*, est donc de recevoir en simultané les messages de tous les clients et de les renvoyer vers tous les autres clients.

Le serveur doit :

- Attendre des connexions,
- Créer un nouveau thread pour chaque nouveau client,
- Créer et maintenir une liste des clients connectés,

Le thread serveur, nommé *ThreadClientIRC*, (chargé d'un seul client) doit :

- Envoyer un message de bienvenu,
- Envoyer la liste de tous les clients connectés,
- Faire passer chaque message provenant du client au serveur, qui à son tour l'envoi à tous les clients connectés.

Pour créer la liste des clients, vous utiliserez la classe java.util.Vector. Elle possède 4 méthodes utiles :

- void addElement(Object obj) : ajoute un objet,
- removeElement(Object obj) : supprime un objet,
- int size( ) : retourne le nombre d'éléments,
- Object elementAt(int index) : retourne l'objet à la position donnée (à l'index).

```
public class ClientIRC extends Thread{
      static int port=…;
      static InetAddress hote=…;
      Socket sc=…;
      BufferedReader in; PrintWriter out;
      String nom;
      int compteur=0;

      public ClientIRC(String nom){
            …
      }
      public void run(){
            try{
                  sc = …
                  in =…
                  out =…
                  String rep; String req;

                  //envoyer le pseudonyme au serveur

                  //recevoir le message d'accueil du serveur

                  for (int i = 0; i < 10; i++) {
                        //recevoir un message du serveur

                        //incrementer le nb d'echanges

                        //repondre au serveur

                        //faire une pause de 3sec
                  }
                  //recevoir un message du serveur

                  //faire une pause de 2sec

                  //envoyer un message « Bye » au serveur

            }
            catch(IOException e){
                  System.err.println("Impossible cree socket du client : " +e);
            }
            finally{
                  try{
                        sc.close();in.close();out.close();
                  }
                  catch (IOException e){}
            }
      }
      public static void main (String[] args){
            try{
                  if (args.length>=2){
                                          …
                  }
                  else{
                        …
                  }
            }
            catch(UnknownHostException e){
                  System.err.println("Machine inconnue :" +e);
            }
            ClientIRC client=…
      }
}
```

```
public class ServeurIRC{
      int port=…
      ServerSocket se=…
      Socket ssv=…
      int cl=…
      Vector V;

      public ServeurIRC(){
            try{
                  V = …
                  se =…

                  while(true){
                        ssv  = …
                        cl++;
                        ThreadClientIRC th=…
                        ajouterClient(th);

                  }
            }
            catch (IOException e){
                  System.err.println("Erreur : " +e);
            }
            finally{
                  try{
                        se.close();
                  }
                  catch (IOException e){}
            }
      }

      public void ajouterClient(ThreadClientIRC c){…}

      synchronized public void supprimerClient(ThreadClientIRC c){…}

      synchronized public void EnvoyerATous(String s){…}

      synchronized public void EnvoyerListeClients(PrintWriter out){…}

      public static void main(String[] args){
            new ServeurIRC();
      }
}
```

```
public class ThreadClientIRC extends Thread{
      ServeurIRC serv;
      Socket ssv;
      String nom;
      BufferedReader in;
      PrintWriter out;

      public ThreadClientIRC(Socket ssv, ServeurIRC serv){…}

      public void run(){
            try{
                  in =…
                  out=…

                  String req = in.readLine();
                  setNom(req);

                  //envoi un premier message d'accueil
                  …
                  //envoi la liste des clients connectes
                  …
                  while(true){
                        //attendre un phrase de reponse
                        …
                        req = in.readLine();
                        if (req.equals("Bye")){
                              //supprimer le client
                              …
                        }
                        //repondre au client
                        …
                        //envoyer le message à tous les clients
                        …
                  }
                  System.out.println (" Fin Communication");
            }
            catch (IOException e){
                  System.err.println("Erreur : " +e);
            }
            finally{
                  try{
                        ssv.close(); in.close(); out.close();
                  }
                  catch (IOException e){}
            }
      }

      public void Envoyer(String s){…}

      public void setNom(String s){…}

      public String getNom(){…}

}
```