TD 1 Introduction aux Applications Distribuées en Java

1. Définition d'un socket

Un socket est un point d'extrémité d'une liaison de communication réseau bidirectionnelle entre un serveur et un client. C'est le couple <Adresse IP, Numéro de port>.

2. Programmation réseau en Java

L'API (Application Programming Interface) client-serveur fourni par le langage Java pour la manipulation des sockets est situé dans le paquetage "java.net".

2.1 Les classes

- **`java.net.InetAddress`** permet de manipuler des adresses IP.
- **`java.net.SocketServer`** permet de programmer le côté serveur en mode connecté (TCP).
- **`java.net.Socket`** permet de programmer le côté client.
- **`java.net.DatagramSocket / java.net.DatagramPacket`** permettent de programmer la communication en mode datagramme ou non connecté (UDP).

2.1.1 Classe java.net.InetAddress

Cette classe représente les adresses IP, ainsi qu'un ensemble de méthodes pour les manipuler :

- **`public static InetAddress getLocalHost() throws UnknownHostException`**
  Donne l'adresse IP de la machine locale.
- **`public static InetAddress getByName(String host) throws UnknownHostException`**
  Donne l'adresse IP de la machine dont le nom est passé en paramètre.
- **`public static InetAddress[] getAllByName(String host) throws UnknownHostException`**
  Permet d'obtenir les différentes adresses IP d'un site.
- **`public String getHostName()`**
  Retourne le nom de la machine dont l'adresse est stockée dans l'objet.

2.1.2 Classe ServerSocket

- **`public ServerSocket(int port) throws IOException`**
  Ce constructeur crée un objet serveur à l'écoute du port spécifié.
- **`public Socket accept() throws IOException`**
  Cette méthode est bloquante, le processus faisant appel à cette méthode se met en attente des demandes de connexion.

2.1.3 Classe java.net.Socket

- **`public Socket(String host, int port) throws UnknownHostException, IOException throws IOException`**
  Ce constructeur construit un socket connecté à la machine et au port spécifiés.

Remarques :

Les deux méthodes suivantes sont utilisées pour obtenir les flux en entrée et en sortie :

- **`public InputStream getInputStream() throws IOException/ public OutputStream getOutputStream() throws IOException`**
  Ces flux sont utilisés par les classes **`java.io.PrintWriter`** et **`java.io.BufferedReader`** pour effectuer l'écriture/lecture.
- **`public void close()`** ferme la connexion et libère les ressources associées au socket.

2.1.4 Classe java.net.DatagramSocket (UDP)

- **`public DatagramSocket(int port) throws SocketException`**
  Ce constructeur construit un socket datagramme (en spécifiant éventuellement un port) sur la machine locale.
- **`public void send(DatagramPacket p) throws IOException/public void receive(DatagramPacket p) throws IOException`**
  Ces opérations permettent d'envoyer et de recevoir un paquet (**`java.net.DatagramPacket`**) qui possède une zone de données, une adresse IP et un numéro de port destinataire .
  Le constructeur du paquet Datagram est sous la forme suivante: **`public DatagramPacket(byte[] buf, int length, InetAddress address, int port)`**
- **`public void connect(InetAddress address, int port)`**
  Permet de connecter un socket datagramme à un destinataire. Dans ce cas, les paquets émis sur le socket seront toujours pour l'adresse spécifiée (il n'est plus nécessaire de spécifier l'adresse de destination pour chacun des paquets).
- **`public void disconnect()`**
  Fait la déconnexion.
- **`public void close()`**
  Elle libère les ressources du système associées au socket.

3. Exemple

Dans cet exemple, vous allez apprendre à créer une application de chat java entre deux machines en utilisant les sockets.

3.1 Serveur

Le *serveur java* initialise la connexion, il lance l'écoute sur un port et se met en attente des connexions entrantes pour qu'il les accepte. Par exemple, le numéro de port est 5000, le client envoie une demande au serveur avec ce numéro de port 5000. Le serveur accepte la demande et transmit ses informations (adresse ip) au client. Maintenant, la connexion est établie et un échange de messages peut se faire.

On a besoin aussi d'un outil pour saisir, envoyer et recevoir le flux:

- Scanner: lire les entrées clavier.
- BufferedReader: lire le texte reçu à partir de l'émetteur.

- PrintWriter: envoyer le texte saisi.

Après la création du socket serveur qui porte le numéro de port 5000, le serveur attend les connexions entrantes et dès qu'une est détectée, il l'accepte. Les deux variables qui gèrent le flux de lecture et d'écriture **`in`** et **`out`** sont initialisées pour qu'elles soient reliées directement avec le flux d'envoi et de réception.

La variable s stocke le texte saisi avec la méthode **`next()`**, puis elle est envoyée avec la méthode **`println(s)`**.

La méthode **`flush()`** est importante car elle vide le buffer d'écriture vers la sortie sinon un **`null`** va être reçu par l'autre machine.

La variable **`message_client`** stocke le message reçu qui est affiché avec **`println().`**

La limite de ce code est qu'il est capable d'envoyer et recevoir un message qu'une seule fois. Vous aurez dans la tête une idée d'ajouter une boucle **`while(true`**). C'est vrai, mais si par exemple le serveur envoie un message au client, ce dernier ne peut pas récupérer le message tant qu'il n'a pas envoyer lui aussi. Pour cela, une solution optimale peut être approchée nous créons deux Threads: un pour l'envoie et l'autre pour la réception. Ces deux processus permettent que l'envoi et la réception se fassent simultanément.

La séparation des deux processus est clair, le Serveur et le Client peuvent échanger les données à tout moment et infiniment. La boucle while de la lecture teste si la connexion est encore établie. N'oubliez pas de bien fermer vos flux de lecture et d'écriture ainsi que la connexion après la sortie de la boucle avec la méthode **`close()`**.

3.2 Client

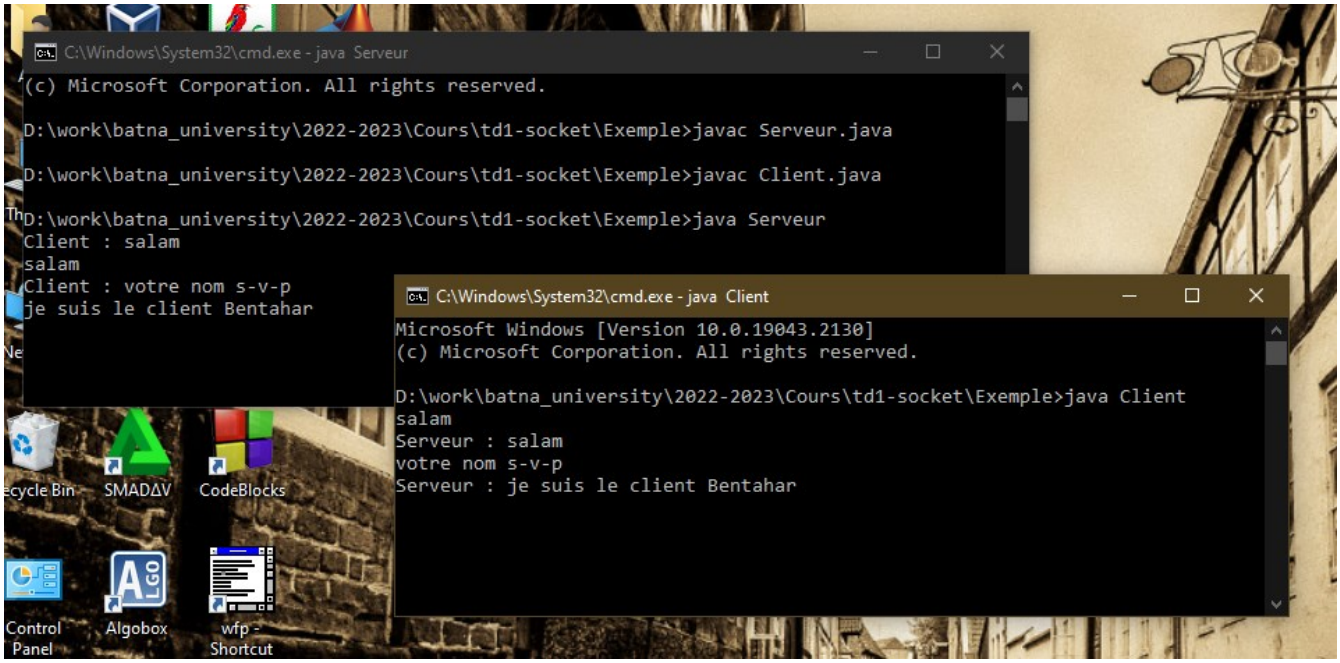
Le client n'a besoin qu'à la classe **`Socket`** pour établir la *connexion serveur*, le constructeur prend en entrée l'adresse IP du serveur et le numéro de port. Le reste du code est le même que celui du serveur.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
public class Serveur {

   public static void main(String[] test) {

     final ServerSocket serveurSocket  ;
     final Socket clientSocket ;
     final BufferedReader in;
     final PrintWriter out;
     final Scanner sc=new Scanner(System.in);

     try {
       serveurSocket = new ServerSocket(5000);
       clientSocket = serveurSocket.accept();
       out = new PrintWriter(clientSocket.getOutputStream());
       in = new BufferedReader (new InputStreamReader
(clientSocket.getInputStream()));
       Thread envoi= new Thread(new Runnable() {
          String msg;
          @Override
          public void run() {
             while(true){
                msg = sc.nextLine();
                out.println(msg);
                out.flush();
             }
          }
       });
       envoi.start();
       Thread recevoir= new Thread(new Runnable() {
          String msg ;
          @Override
          public void run() {
             try {
                msg = in.readLine();
                //tant que le client est connecté
                while(msg!=null){
                   System.out.println("Client : "+msg);
                   msg = in.readLine();
                }
                //sortir de la boucle si le client a déconecté
                System.out.println("Client déconecté");
                //fermer le flux et la session socket
                out.close();
                clientSocket.close();
                serveurSocket.close();
             } catch (IOException e) {
                  e.printStackTrace();
             }
          }
       });
       recevoir.start();
       }catch (IOException e) {
          e.printStackTrace();
       }
   }
}
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
/*
 * www.codeurjava.com
 */
public class Client {

   public static void main(String[] args) {

      final Socket clientSocket;
      final BufferedReader in;
      final PrintWriter out;
      final Scanner sc = new Scanner(System.in);//pour lire à partir du clavier

      try {
         /*
         * les informations du serveur ( port et adresse IP ou nom d'hote
         * 127.0.0.1 est l'adresse local de la machine
         */
         clientSocket = new Socket("127.0.0.1",5000);

         //flux pour envoyer
         out = new PrintWriter(clientSocket.getOutputStream());
         //flux pour recevoir
         in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

         Thread envoyer = new Thread(new Runnable() {
             String msg;
              @Override
              public void run() {
                while(true){
                  msg = sc.nextLine();
                  out.println(msg);
                  out.flush();
                }
             }
         });
         envoyer.start();

        Thread recevoir = new Thread(new Runnable() {
            String msg;
            @Override
            public void run() {
               try {
                 msg = in.readLine();
                 while(msg!=null){
                    System.out.println("Serveur : "+msg);
                    msg = in.readLine();
                 }
                 System.out.println("Serveur déconecté");
                 out.close();
                 clientSocket.close();
               } catch (IOException e) {
                   e.printStackTrace();
               }
            }
        });
        recevoir.start();

      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

s

4. Exécution :



5. Références :

http://www.codeurjava.com/2014/11/java-socket-clientserveur-mini-chat.html

Suivez la vidéo YouTube suivante:

https://www.youtube.com/watch?v=d-eD6EDa3io

6. TP 1 à domicile

Faire un programme montre une connexion entre un processus serveur et client (mode connecté TCP). Les deux processus s'échangent des messages 10 fois, puis le client envoie un message de terminaison.

Remarque**: Le TP à faire à domicile et à rendre à moi sur bentahar.info@aol.com avant le **27/10/2022