Utilisation des sockets sous android

Plan

I. Runnable

- Cette interface doit être implémentée par toute classe qui contiendra des traitements à exécuter dans un thread.
- Cette interface ne définit qu'une seule méthode : void run().
- Dans les classes qui implémentent cette interface, la méthode run() doit être redéfinie pour contenir le code des traitements qui seront exécutés dans le thread. [1]

```
Runnable r = new Runnable() {
        @Override
        public void run() {
        }
};
```

II. Thread

La classe Thread est définie dans le package java.lang. Elle implémente l'interface Runnable.

Constructeur	**Rôle**
Thread()	Créer une nouvelle instance
Thread(Runnable target)	Créer une nouvelle instance en précisant les traitements à exécuter

Méthode	**Rôle**
static void sleep(long millis)	Endormir le thread pour le délai exprimé en millisecondes précisé en paramètre
void start()	Lancer l'exécution des traitements : associer des ressources systèmes pour l'exécution et invoquer la méthode run()

III. Utilisation du protocole TCP [1]

TCP est un protocole qui permet une connexion de type point à point entre deux applications. C'est un protocole fiable qui garantit la réception dans l'ordre d'envoi des données. En contrepartie, ce protocole offre de moins bonnes performances mais c'est le prix à payer pour la fiabilité.

TCP utilise la notion de port pour permettre à plusieurs applications d'exploiter ce même protocole.

Dans une liaison entre deux ordinateurs, l'un des deux joue le rôle de serveur et l'autre celui de client.

III.1. La classe ServerSocket [1]

La classe ServerSocket est utilisée côté serveur : elle attend simplement les appels du ou des clients. C'est un objet du type Socket qui prend en charge la transmission des données.

Cette classe représente la partie serveur du socket. Un objet de cette classe est associé à un port sur lequel il va attendre les connexions d'un client. Généralement, à l'arrivée d'une demande de connexion, un thread est lancé pour assurer le dialogue avec le client sans bloquer les connexions des autres clients.

La classe SocketServer possède plusieurs constructeurs dont les principaux sont :

Constructeur	Rôle
ServerSocket()	Constructeur par défaut
ServerSocket(int)	Créer un socket sur le port fourni en paramètre
ServerSocket(int, int)	Créer un socket sur le port et avec la taille maximale de la file fournis en paramètres

Tous ces constructeurs peuvent lever une exception de type IOException.

La classe SocketServer possède plusieurs méthodes :

Méthode	Rôle
Socket accept()	Attendre une nouvelle connexion
void close()	Fermer la socket

Si un client tente de communiquer avec le serveur, la méthode accept() renvoie une socket qui encapsule la communication avec ce client.

La mise en œuvre de la classe SocketServer suit toujours la même logique :

- créer une instance de la classe SocketServer en précisant le port en paramètre
- définir une boucle sans fin contenant les actions ci-dessous
- appelle de la méthode accept() qui renvoie une socket lors d'une nouvelle connexion
- obtenir un flux en entrée et en sortie à partir de la socket
- écrire les traitements à réaliser

III.2. La classe Socket [1]

Les sockets implémentent le protocole TCP (Transmission Control Protocol). La classe contient les méthodes de création des flux d'entrée et de sortie correspondants. Les sockets constituent la base des communications par le réseau.

Comme les flux Java sont transformés en format TCP/IP, il est possible de communiquer avec l'ensemble des ordinateurs qui utilisent ce même protocole. La seule condition importante au niveau du système d'exploitation est qu'il soit capable de gérer ce protocole.

Cette classe encapsule la connexion à une machine distante par le réseau. Elle gère la connexion, l'envoi de données, la réception de données et la déconnexion.

La classe Socket possède plusieurs constructeurs dont les principaux sont :

Constructeur	Rôle
Server()	Constructeur par défaut
Server(String, int)	Créer une socket sur la machine dont le nom et le port sont fournis en paramètres
Server(InetAddress, int)	Créer une socket sur la machine dont l'adresse IP et le port sont fournis en paramètres

La classe Socket possède de nombreuses méthodes dont les principales sont :

Méthode	Rôle
InetAddress getInetAddress()	Renvoie l'adresse I.P. à laquelle la socket est connectée
void close()	Fermer la socket
InputStream getInputStream()	Renvoie un flux en entrée pour recevoir les données de la socket
OutputStream getOutputStream()	Renvoie un flux en sortie pour émettre les données de la socket
int getPort()	Renvoie le port utilisé par la socket

La mise en œuvre de la classe Socket suit toujours la même logique :

- créer une instance de la classe Socket en précisant la machine et le port en paramètres
- obtenir un flux en entrée et en sortie
- écrire les traitements à réaliser

III.3. La classe Inet4Address [2]

Cette classe permet de manipuler les adresses Internet.

Il n'existe pas de constructeur. Pour obtenir une instance de cette classe, il est nécessaire d'utiliser des méthodes de classe.

Ces méthodes de classe sont les suivantes :

- public static InetAddress getLocalHost() throws UnknownHostExeception
  Retourne un objet contenant l'adresse Internet de la machine locale.
- public static synchronized InetAddress getByName(String host_name) throws UnknownHostExeception
  Retourne un objet contenant l'adressse Internet de la machine dont le nom est passé en paramètre.

Les méthodes que l'on peut appliquer à un objet de cette classe sont :

- public String getHostName ()
  Retourne le nom de la machine dont l'adresse est stockée dans l'objet.
- public byte[] getAddress ()
  Retourne l'adresse internet stockée dans l'objet sous forme d'un tableau de 4 octets dans l'ordre réseau.
- public String toString ()
  Retourne une chaîne de caractères qui liste le nom de la machine et son adresse.

Exemple :

```
InetAddress adresseLocale = InetAdress.getLocalHost ();
InetAddress adresseServeur = InetAdress.getByName("www.univ-mlv.fr");
System.out.println(adresse);
```

III.4. Lecture des données (réception)

La lecture peut utiliser les classes InputStream, InputStreamReader et BufferedReader.

```
private BufferedReader br;
…
br = new BufferedReader(new InputStreamReader(s.getInputStream()));
String cmd = br.readLine();
```

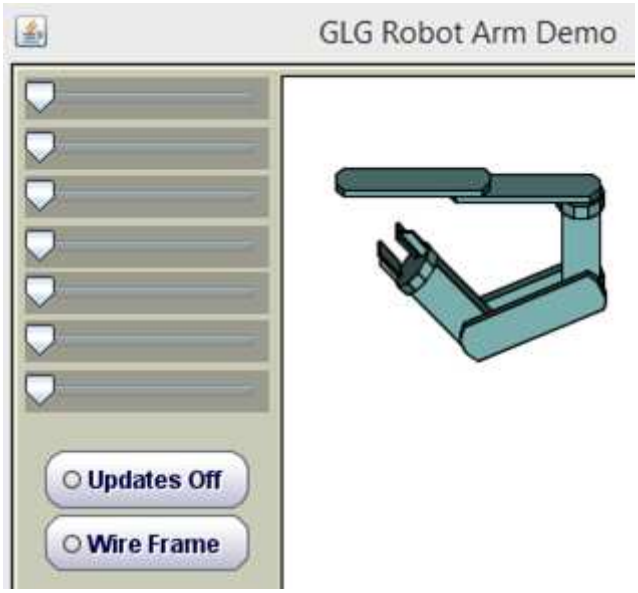
III.5. Ecriture des données (envoi)

L'écriture peut utiliser les classes OutputStream, OutputStreamWriter, BufferedWriter et PrintWriter

```
private PrintWriter pw;
…
pw = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(s.getOutputStream())), true);
String message ="Test envoi" ;
pw.println(message);
```

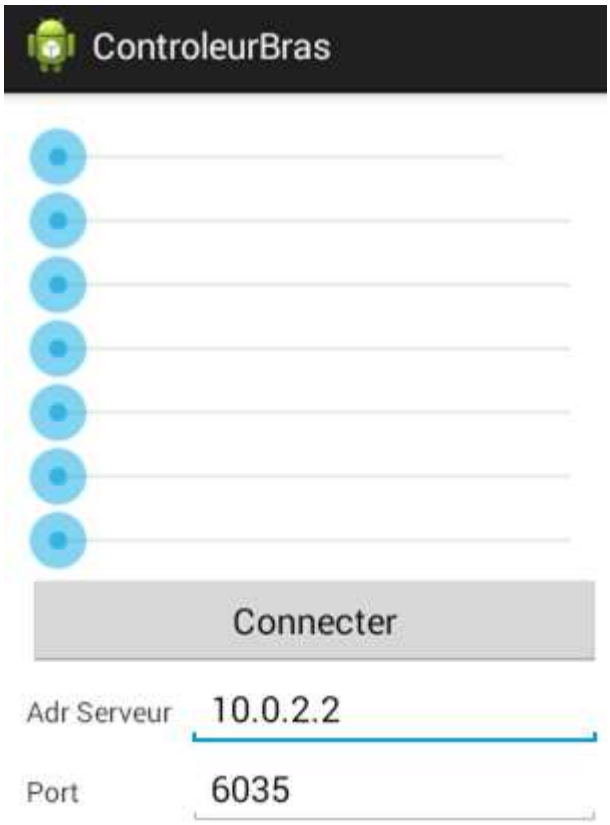
Serveur : Bras

```
package bras;
public class GlgRobotArmDemo extends GlgJBean implements ActionListener {
        private static JSlider[] tSlider;
    private static int indice;
    private static final int PORT = 6035;
    private ServerSocket ss;
    private Socket s;
    private BufferedReader br;
```




```
    public GlgRobotArmDemo() {
        super();
        SetDResource("$config/GlgAntiAliasing", 1.0); // Enable antialiasing
        lancerThreadServeur();
    }
    private void lancerThreadServeur() {
        Runnable r = new Runnable() {
            @Override
            public void run() {
                demmarerServeur();
            }
        };
        Thread th = new Thread(r);
        th.start();
    }
    private void demmarerServeur() {
        try {
            ss = new ServerSocket(PORT);
            s = ss.accept();
            br = new BufferedReader(new InputStreamReader(s.getInputStream()));
            while (true) {
                String cmd;
                cmd = br.readLine();
                execterCommande(cmd);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private void execterCommande(String cmd) {
        System.out.println(cmd);
        String[] t = cmd.split(":");
        if (t.length > 1) {
            int indice = Integer.parseInt(t[0]);
            int val = Integer.parseInt(t[1]);
            tSlider[indice].setValue(val);
        }
    }
}
```

Client : ControleurBras




```
package com.bras;
public class MainActivity extends Activity {
    private SeekBar[] tSeek;
    private Button btnConnecter;
    public static final int PORT_SERVEUR = 6035;
    public static final String ADR_SERVEUR = "10.0.2.2";
    private Socket s;
    private PrintWriter pw;
    private EditText edAdresse;
    private EditText edPort;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        init();
    }
    private void init() {
        tSeek = new SeekBar[7];
        tSeek[6] = (SeekBar) findViewById(R.id.seekBar);
        tSeek[5] = (SeekBar) findViewById(R.id.seekBar1);
        tSeek[4] = (SeekBar) findViewById(R.id.seekBar2);
        tSeek[3] = (SeekBar) findViewById(R.id.seekBar3);
        tSeek[2] = (SeekBar) findViewById(R.id.seekBar4);
        tSeek[1] = (SeekBar) findViewById(R.id.seekBar5);
        tSeek[0] = (SeekBar) findViewById(R.id.seekBar6);
        btnConnecter = (Button) findViewById(R.id.btnConnecter);
        edAdresse = (EditText) findViewById(R.id.edAdresse);
        edPort = (EditText) findViewById(R.id.edPort);
        edPort.setText(PORT_SERVEUR + "");
        edAdresse.setText(ADR_SERVEUR + "");
        ajouterEcoteur();
    }
    private void ajouterEcoteur() {
        btnConnecter.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                lanceThreadClient();
            }
        });
        for (int j = 0; j < tSeek.length; j++) {
            final int finalJ = j;
            tSeek[j].setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
                @Override
                public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
                    envoyer(finalJ + ":" +  seekBar.getProgress());
                }

                @Override
                public void onStartTrackingTouch(SeekBar seekBar) {

                }

                @Override
                public void onStopTrackingTouch(SeekBar seekBar) {

                }
            });
        }
}
```

```
 private void lanceThreadClient() {
      Runnable r = new Runnable() {
         @Override
         public void run() {
            demmarerClient();
         }

      };
      Thread th = new Thread(r);
      th.start();
      btnConnecter.setVisibility(View.GONE);
   }

   private void demmarerClient() {
      try {
         Inet4Address i = (Inet4Address) Inet4Address.getByName(edAdresse.getText().toString());
         s = new Socket(i, Integer.parseInt(edPort.getText().toString()));
         pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(s.getOutputStream())), true);
      } catch (UnknownHostException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }

   }

   private void envoyer(final String cmd) {
      Runnable r = new Runnable() {
         @Override
         public void run() {
            if (pw != null) {
               pw.println(cmd);
               pw.flush();
            }
         }
      };
      Thread th = new Thread(r);
      th.start();

   }
}
```

Références :

[1] https://www.jmdoudoux.fr/java/dej/chap-net.htm
[2] https://www.irif.fr/~sighirea//cours/reseauxM/java.inet.html
[3] https://www.jmdoudoux.fr/java/dej/chap-serialisation.htm