Sareko programazioa

TCP

TCP por sus siglas en inglés: Transmission Control Protocol. Tiene las siguientes características:

- Orientado a conexión
- Entrega garantizada
- Orden de entrega
- Detección de error y corrección
- Control de flow

Los programas de TCP tienen dos archivos: cliente y servidor.

Cliente

Lo primero de todo, es definir la dirección y el puerto del servidor con el que queremos establecer la conexión:

```
final String serverAddress = "192.168.65.X";
// InetAddress serverAddress = InetAddress.getLocalHost();
final int port = 12345;
```

O se lo podemos pedir al usuario:

```
final String serverAddress;
final int port;
Scanner connection = new Scanner(System.in);
System.out.print("Write the ip of the server: ");
serverAddress = connection.nextLine();
System.out.print("Write the port of the server: ");
port = connection.nextInt();
```

El siguiente paso es intentar crear el socket para abrir la conexión con el servidor:

```
try (Socket socket = new Socket(serverAddress, port)) {
      El resto del programa...
} catch (IOException e) {
      e.printStackTrace();
}
```

Después, tenemos que crear los medios para escribir y leer mensajes. Los medios varían en base al tipo de mensaje:

Para datos primitivos:

```
//DataInputStream (Leer o recibir)
InputStream inputStream = socket.getInputStream();
DataInputStream dataInput = new DataInputStream(inputStream);
int intValue = dataInput.readInt();

//DataOutputStream (Escribir o enviar)
OutputStream outputStream = socket.getOutputStream();
DataOutputStream dataOutput = new DataOutputStream(outputStream);
dataOutput.writeInt(42);
dataOutput.flush();
```

Para texto:

```
//BufferedReader (Leer o recibir)
InputStream inputStream = socket.getInputStream();
BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
    // Process the line of text
}

//BufferedWriter (Escribir o enviar)
OutputStream outputStream = socket.getOutputStream();
```

```
BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(outputStream));
writer.write("Hello, server!");
writer.newLine(); // Write a newline to indicate the end of the message
writer.flush(); // Flush the buffer to send the data
```

Para objetos:

```
//ObjectInputStream (Leer o recibir)
InputStream inputStream = socket.getInputStream();
ObjectInputStream objectInput = new ObjectInputStream(inputStream);
MyObject receivedObject = (MyObject) objectInput.readObject();

//ObjectOutputStream (Escribir o enviar)
OutputStream outputStream = socket.getOutputStream();
ObjectOutputStream objectOutput = new ObjectOutputStream(outputStream);
objectOutput.writeObject(myObject);
objectOutput.flush();
```

Servidor

Lo que tenemos que hacer, es definir el puerto desde el que esperamos una conexión, definir el socket del servidor en base a ese puerto, esperar a que se reciba una solicitud de conexión, y cuando llegue aceptarla guardando el socket del servidor. El servidor utiliza los mismos medios de comunicación que el cliente. Por último, hay que cerrar el socket del servidor:

```
final int port = 12345;

try {
    InetAddress localHost = InetAddress.getLocalHost();
    System.out.println("Server is waiting for a connection on "
       + localHost.getHostAddress() + ":" + port);
    ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    System.out.println("Client connected from: "
       + clientSocket.getInetAddress());
    . . .
    clientSocket.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

UDP

UDP por sus siglas en inglés: User Datagram Protocol. Tiene las siguientes características:

- No orientado a conexión
- No garantiza la entrega
- No garantiza el orden
- Procesamiento mínimo de datos
- Simple y ligero
- Broadcast y Multicast

Los programas de UDP tienen dos archivos: cliente y servidor.

Cliente

Lo primero que hay que hacer es definir la dirección y el puerto del servidor y crear el DatagramSocket:

```
final int serverPort = 12345;
    try {
       InetAddress serverAddress =
             InetAddress.getByName("localhost");//Server on Localhost
       DatagramSocket socket = new DatagramSocket();
       . . .
```

Después ya podemos proceder a enviar o a recibir mensajes haciendo conversiones a y desde byte[]:

```
// Enviar el mensaje
String message = "Hello, server!";
byte[] data = message.getBytes();
DatagramPacket packet = new DatagramPacket(data,
data.length,serverAddress,
serverPort);
socket.send(packet);

// Recibir el mensaje
byte[] buffer = new byte[1024];
DatagramPacket responsePacket = new DatagramPacket(buffer,
buffer.length);
socket.receive(responsePacket);
```

```
String serverResponse = new String(responsePacket.getData(), 0,
                                   responsePacket.getLength());
System.out.println("Server says: " + serverResponse);
```

Para enviar y recibir numeros:

```
// Convertir int en byte[]
Scanner in = new Scanner(System.in);
ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
DataOutputStream dataOutput = new DataOutputStream(byteStream);
System.out.println("Sartu mezua:");
int message = in.nextInt();
dataOutput.writeInt(message);
byte[] data = byteStream.toByteArray();

// Recibir byte[] como Integer
byte[] buffer = new byte[1024];
DatagramPacket receivedPacket = new DatagramPacket(buffer,
buffer.length);
socket.receive(receivedPacket);
Integer
receivedNumber=ByteBuffer.wrap(receivedPacket.getData()).getInt();
```

Para enviar y recibir objetos (la clase del objeto tiene que implementar la clase serializable):

```
// Enviar el objeto
Ikaslea ikaslea = new Ikaslea("izena", "abizena", 20, "DAM", true);
objectStream.writeObject(ikaslea);
byte[] data = byteStream.toByteArray();
DatagramPacket packet = new DatagramPacket(data, data.length, address,
port);
socketCliente.send(packet);

// Recibir el objeto
byte[] buffer = new byte[2048];
DatagramPacket receivedPacket = new DatagramPacket(buffer,
buffer.length);
socketServidor.receive(receivedPacket);
// Deserialize the object
Ikaslea ikaslea;
try (ObjectInputStream objectInput = new ObjectInputStream(new
                  ByteArrayInputStream(receivedPacket.getData()))) {
                ikaslea = (Ikaslea) objectInput.readObject();
```

```
}
```

Por último, hay que cerrar la conexión:

```
socket.close();
```

Servidor

Lo único en lo que se diferencia el servidor del cliente el que al crear el DatagramSocket en el servidor, hay que pasarle el puerto por el que espera la conexión:

```
final int port = 12345;
try {
     DatagramSocket socket = new DatagramSocket(port);
     . . .
```

TCP Multithreading

En este tipo de programas, hay tres archivos: Cliente, Servidor, y ServerService.

Cliente

El cliente no cambia respecto a un programa TCP normal.

Servidor

El servidor lo único que hace es empezar un hilo por cada conexión aceptada:

```
ServerSocket servidor = new ServerSocket(6000);

while (true) {
     Socket cliente = servidor.accept();
     ServerService hilo = new ServerService(cliente);
     hilo.start();
}
```

ServerService

Este programa es el hilo y es donde se aplica la lógica para cada cliente. La clase tiene que extender la clase Threads, definir sus propios atributos, y su

constructor. La lógica del programa va en el metodo @Override-ado run(). Después el programa dentro de run() es como un servidor TCP normal.