3.5.1. GESTION DE SOCKETS TCP

El modelo de sockets más simple se muestra en la Figura 3.5:

■ El programa servidor crea un socket de servidor definiendo un puerto, mediante el método ServerSocket(port), y espera mediante el método accept() a que el cliente solicite la conexión

■ Cuando el cliente solicita una conexión, el servidor abrir la conexión al socket con el método accept().

■ El cliente establece una conexión con la maquina host a través del puerto especificado mediante el método Socket(host, port).

■ El cliente y el servidor se comunican con manejadores InputStream y OutputStream. El cliente escribe los mensajes en el OutputStream asociado al socket y el servidor leerá los mensajes del cliente del InputStream. Igualmente, el servidor escribirá los mensajes al OutputStream y el cliente los leerá del InputStream.

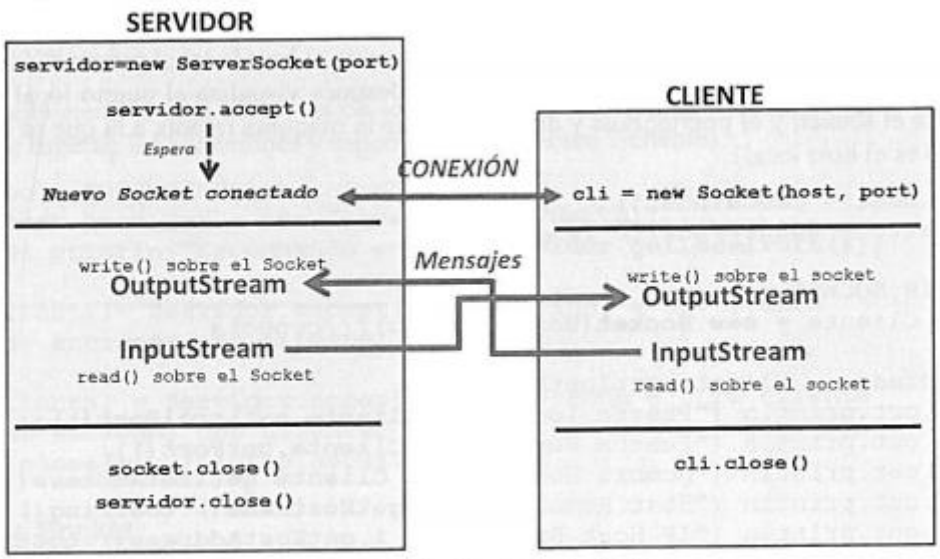

Figura 3.5. Modelo de Socket TCP.

Apertura de sockets.

En el programa servidor se crea un objeto ServerSocket invocando al método ServerSocket() en el que indicamos el número de puerto por el que el servidor escucha las peticiones de conexión de los clientes (se considera el tratamiento de excepciones):

```
ServerSocket servidor=null;
try {
servidor = new ServerSocket(numeroPuerto);
} catch (IOException io){
io.printStackTrace();
}
```

Necesitamos también crear un objeto Socket desde el ServerSocket para aceptar las conexiones, se usa el metodo accept();

Socket clienteConectado=null;

```
try {
clienteConectado = servidor.accept();
```

```
} catch (IoException io) {
io.printStackTrace();
}
```

En el programa cliente es necesario crear un objeto Socket; el socket se abre de la siguiente manera:

```
Socket cliente;
try {
cliente = new Socket("maquina", numeroPuerto);
} catch (I0Exception io) {
io.printStackTrace();
}
```

Donde máquina es el nombre de la maquina a la que nos queremos conectar y numeroPuerto es el puerto por el que el programa servidor está escuchando las peticiones de los clientes.

Hay puertos TCP de 0 a 65535. Los puertos en el rango de 0 a 1023 están reservados para servicios privilegiados; otros puertos de 1024 a 49151 están reservados para aplicaciones concretas (por ejemplo el 330610 usa MySQL, el 1521 Oracle); por Ultimo de 49152 a 65535 no están reservados para ninguna aplicación concreta.

Creación de streams de entrada.

En el programa servidor podemos usar DataInputStream para recuperar los mensajes que el cliente escriba en el socket, previamente hay que usar el método getInputStream0 para obtener el flujo de entrada del socket del cliente:

```
InputStream entrada=null;
try {
entrada = clienteConectado.getInputStream();
} catch (I0Exception e) {
e.printStackTrace();
}
DataInputStream flujoEntrada = new DataInputStream(entrada);
```

En el programa cliente podemos realizar la misma operación para recibir los mensajes procedentes del programa servidor.

La clase DataInputStream permite la lectura de líneas de texto y tipos primitivos Java. Algunos de sus métodos son: readInt(), readDouble(), readLine(), readUTF(), etc.

Creación de streams de salida.

En el programa servidor podemos usar DataOutputStream para escribir los mensajes que queramos que el cliente reciba, previamente hay que usar el método getOutputStream() para obtener el flujo de salida del socket del cliente:

```
OutputStream salida=null;
try {
salida = clienteConectado.getOutputStream();
} catch (I0Exception e1){
ei.printStackTrace();
}
DataOutputStream flujoSalida = new DataOutputStream(salida);
```

En el programa cliente podemos realizar la misma operación para enviar mensajes al programa servidor.

La clase DataOutputStream dispone de metodos para escribir tipos primitivos Java: writeInt(), writeDouble(), writeBytes(), writeUTF(), etc.

Cierre de sockets

El orden de cierre de los sockets es relevante, primero se han de cerrar los streams relacionados con un socket antes que el propio socket:

```
try {
entrada.close();
flujoEntrada.close();
salida.close();
flujoSalida.close();
clienteConectado.close();
servidor.close();
} catch (IOException e) {
e.printStackTrace();
}
```

A continuación, se muestra un ejemplo de un programa servidor que recibe un mensaje de un cliente y lo muestra por pantalla; después envía un mensaje al cliente. Se han eliminado los bloques try-catch para que el código resulte más legible:

```
import java.io.*;
import java.net.*;

public class TCPejemplolServidor {
public static void main(String[] arg) throws lOException {
int numeroPuerto = 6000;// Puerto
ServerSocket servidor = new ServerSocket(numeroPuerto);
Socket clienteConectado = null;
System.out.println("Esperando al cliente ");
ClienteConectado = servidor.accept();

// CREO FLUJO DE ENTRADA DEL CLIENTE
InputStream entrada = null;
entrada = clienteConectado.getInputStream();
DataInputStream flujoEntrada - new DataInputStream(entrada);

// EL CLIENTE ME ENVIA UN MENSAJE
System.out.println("Recibiendo del CLIENTE: \n\t" +
flujoEntrada.readUTF());

// CREO FLUJO DE SALIDA AL CLIENTE
OutputStream salida = null;
salida = clienteConectado.getOutputStream();
DataOutputStream flujoSalida = new DataOutputStream(salida);

// ENVIO UN SALUDO AL CLIENTE
flujoSalida.writeUTF("Saludos al cliente del servidor");

// CERRAR STREAMS Y SOCKETS
entrada.close();
flujoEntrada.close(); salida.close();
flujoSalida.close();
clienteConectado.close();
servidor.close();
}// main
}// fin
```

El programa cliente, en primer lugar envía un mensaje al servidor y despues recibe un mensaje del servidor visualizándolo en pantalla, se ha simplificado la obtención de los flujos de entrada y salida:

```
import java.io.*;
import.java.net.*;
public class TCPejemplo1Cliente{
static void main(String[] args) throws Exception{-
String Host =  "localhost";
int Puerto =  6000;//puerto remoto

System.out.println{"PROGRAMA CLIENTE INICIADO….");
Socket Cliente = new Socket(Host, Puerto);

//  CREO FLUJO DE SALIDA AL SERVIDOR :
DataoutputStream flujoSalida =  new
DataOutputStream(Cliente.getOutputStream());

// ENVIO UN SALUDO AL SERVIDOR
flujoSalida.writeUTF("Saludos al SERVIDOR DESDE EL CLIENTE");

//  CREO FLUJO DE ENTRADA AL SERVIDOR
DataInputStream flujoEntrada          =  new
DataInputStream(Cliente.getInputStream());

// EL  SERVIDOR ME ENVIA UN MENSAJE
System.out.println("Recibiendo del SERVIDOR:   \n\t"       +
flujoEntrada.readUTF());

//  CERRAR STREAMS Y SOCKETS
flujoEntrada.close();
fLujoSalida.close();
Cliente.Close();
}//main
}//
```

ACTIVIDAD 3.4

Crea un programa servidor que envíe un mensaje a otro programa cliente y el programa cliente que le devuelva el mensaje en minúscula.

ACTIVIDAD 3.5.1

Crea un programa cliente que introduzca por teclado un número entero y se lo envíe al servidor. El servidor le devolverá el cuadrado del número.

ACTIVIDAD 3.5.2

Crea un programa servidor que pueda atender hasta 3 clientes. Debe enviar a cada cliente un mensaje indicando el número de cliente que es. Este número será 1, 2 o 3. El cliente mostrará el mensaje recibido. Cambia el programa para que lo haga para N clientes, siendo N un parámetro que tendrás que definir en el programa.