

# Práctica 2

## Uso de clases java sockets TCP y UDP



Enlace de carpeta Google Drive:

[https://drive.google.com/drive/folders/1j6qQOzS2T8oFlsQaOZm7DfuVv4TbTq15?usp=drive\\_link](https://drive.google.com/drive/folders/1j6qQOzS2T8oFlsQaOZm7DfuVv4TbTq15?usp=drive_link)

GitHub:

[https://github.com/Archerd6/Practica\\_2\\_Netes\\_Cliente\\_Servidor](https://github.com/Archerd6/Practica_2_Netes_Cliente_Servidor)

## ***TAREA 1: Servidor y Cliente TCP***

La aplicación permite al usuario introducir una cadena desde el teclado, la cual se envía a través de un socket TCP al servidor. El servidor devuelve esa cadena introducida anteriormente revertida y en mayúsculas al cliente, que muestra por pantalla. Esta interacción se inicia después de establecer una conexión entre el cliente y el servidor, que se cierra tras escribir por teclado "END" que tras recibirla el servidor, este responde con "OK".

En el servidor, se indica la dirección IP y el puerto del cliente que se ha conectado. Además, se extraen los datos que el cliente envía para su procesamiento. Ambos lados, cliente y servidor, utilizan flujos de entrada y salida para la comunicación a través de sockets.

## Servidor TCP

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

class ServerTCP
{
    @SuppressWarnings("resource")
    public static void main(String[] args) throws IOException
    {
        ServerSocket server = null;
        Socket client = null;
        PrintWriter out = null;
        Scanner in = null;
        String line;
        int port = 12345; // Puerto del servidor

        /*
         * COMPLETAR Crear e inicializar el socket del servidor
         */
        try
        {
            server = new ServerSocket(port);
        }
        catch (IOException e)
        {
            System.out.println("Could not listen on port " + port);
            System.exit(-1);
        }

        while (true) // Funcion del servidor:
        {
            System.out.println("Waiting for a new TCP client");
            try
            {
                /*
                 * COMPLETAR Esperar conexiones entrantes */
                client = server.accept();
                System.out.println("Connecting with: " + client.getInetAddress().getHostAddress());
                System.out.println("port: " + client.getPort());
            }
            catch (IOException e)
            {
                System.out.println("Accept failed: " + port);
                System.exit(-1);
            }

            try
            {
                /*
                 * COMPLETAR Una vez aceptada una conexion, inicializar flujos de entrada/salida
                 * del socket conectado
                 */
                in = new Scanner(client.getInputStream());
                out = new PrintWriter(client.getOutputStream());
            }
            catch (IOException e)
            {
                System.out.println("Exception " + e);
                System.exit(-1);
            }

            boolean salir = false;
            while (!salir) /* Inicio bucle del servicio de Eco (1 cliente) */
            {
                try
                {
                    /*
                     * COMPLETAR Recibir texto en variable String line enviado por el cliente
                     * a traves del flujo de entrada del socket conectado
                     */
                    line = in.nextLine();
                    System.out.println("Received from client " + line);

                    /* COMPLETAR Comprueba si es fin de conexion */
                    if (line.compareTo("END") != 0) {
                        /*
                         * COMPLETAR Revertir y poner en mayúscula la cadena y Enviar texto al cliente
                         * a traves del flujo de salida del socket conectado
                         */
                        String revertedLine = new StringBuffer(line.toUpperCase()).reverse().toString();
                        out.println(revertedLine);
                        System.out.println("Sending to client " + revertedLine);
                        out.flush();
                    }
                    else // El cliente quiere cerrar conexion, ha enviado END
                    {
                        /* COMPLETAR Envía OK al cliente */
                        out.print("OK");
                        out.flush();
                        salir = true;
                    }
                }
                catch (Exception e)
                {
                    System.out.println("Read failed");
                    System.exit(-1);
                }
            } // fin del servicio

            System.out.println("Closing connection with the client");
            /* COMPLETAR Cerrar flujos y socket */
            in.close();
            out.close();
            client.close();
        } // fin del bucle
    }
}
```

## Cliente TCP

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

public class ClientTCP
{
    public static void main(String[] args) throws IOException
    {
        String serverName = "localhost";
        int portNumber = 12345;
        Socket serviceSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

        try
        {
            /* COMPLETAR Crear socket y conectar con servidor */
            InetAddress serverAddr = InetAddress.getByName(serverName);
            serviceSocket = new Socket(serverAddr, portNumber);
            System.out.println("Conexion local"+serverAddr);

            /*COMPLETAR Inicializar los flujos de entrada/salida del socket conectado en las variables PrintWriter y BufferedReader*/
            in = new BufferedReader(new InputStreamReader(serviceSocket.getInputStream()));
            out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(serviceSocket.getOutputStream())), true);
        }
        catch (UnknownHostException e)
        {
            System.err.println("Unknown: " + serverName);
            System.exit(1);
        }
        catch (IOException e)
        {
            System.err.println("Couldn't get I/O for " + "the connection to: " + serverName);
            System.exit(1);
        }

        System.out.println("STATUS: Conectado al servidor ");
        System.out.println("STATUS: El puerto (cliente) que se ha usado es: " + serviceSocket.getLocalPort()); // Puerto local
        /* Obtener texto por teclado */
        String userInput;
        System.out.println("Introduzca un texto a enviar (END para acabar)");

        /* Enviar texto leído y almacenado en userInput al servidor a través del socket */
        userInput = stdIn.readLine();

        /* COMPLETAR Comprobar si el usuario ha iniciado el fin de la interaccion */
        while (userInput.compareTo("END") != 0)
        {
            /* COMPLETAR Enviar texto en userInput al servidor a través del flujo de salida del socket conectado*/
            out.println(userInput);

            System.out.println("STATUS: Enviando " + userInput); // Muestra por pantalla el texto enviado

            System.out.println("STATUS: Esperando eco"); // Muestra por pantalla estado

            /* COMPLETAR Recibir texto en echo enviado por el servidor a través del flujo de entrada del socket conectado */
            String echo = null;

            echo = in.readLine(); /* Leer línea del socket*/

            System.out.println("echo: " + echo); // Muestra por pantalla el eco recibido

            /* Leer texto de usuario por teclado */
            System.out.println("Introduzca un texto a enviar (END para acabar)");
            userInput = stdIn.readLine();
        } /* Fin del bucle de servicio en cliente */

        // Salimos porque el cliente quiere terminar la interaccion, ha introducido END
        System.out.println("STATUS: El cliente quiere terminar el servicio");

        /* COMPLETAR Enviar END al servidor para indicar el fin del servicio */
        out.println(userInput);

        System.out.println("STATUS: Sending " + userInput); // Muestra por pantalla el texto enviado
        System.out.println("STATUS: Waiting for the reply"); // Muestra por pantalla el estado

        /* COMPLETAR Recibir OK enviado por el servidor confirmando EL FIN DEL SERVICIO */
        String ok = in.readLine(); // Leer línea del socket

        System.out.println("STATUS: Cerrando conexion " + ok); // Muestra por pantalla el eco recibido
        /* COMPLETAR Cerrar flujos y socket */
        out.close();
        in.close();
        stdIn.close();
        serviceSocket.close();

        System.out.println("STATUS: Conexion cerrada");
    }
}
```

## TAREA 2: Servidor y Cliente UDP

En UDP, esta interacción finalizará cuando el usuario introduzca por teclado la cadena "END", que no enviará al servidor. A continuación, el cliente finalizará. El servidor seguirá activo.

### Servidor UDP

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class ServerUDP
{
    @SuppressWarnings("resource")
    public static void main(String[] args) throws IOException
    {
        DatagramSocket server = null;
        String line;
        int port = 54322; //puerto del servidor
        try
        {
            /*
             * Crear e inicializar el socket del servidor
             */
            server = new DatagramSocket(port);
        }
        catch (IOException e)
        {
            System.out.println("Could not listen on port "+port);
            System.exit(-1);
        }

        // funcion PRINCIPAL del servidor:
        while (true)
        {
            System.out.println("Waiting for a new UDP client");

            /* Crear e inicializar un datagrama VACIO para recibir */
            DatagramPacket packet = new DatagramPacket(new byte[255], 255);

            /* Recibir datagrama */
            server.receive(packet);

            /* Mostrar por pantalla la direccion socket del cliente que solicito el servicio de eco */
            System.out.println("IP cliente: " + packet.getAddress().getHostAddress() +
                               " Puerto cliente: " + packet.getPort());

            /* revertir y poner en mayúscula la cadena y Crear datagrama de respuesta */
            String lectura = new String(packet.getData(), 0, packet.getLength());
            String cadenaMayuscula = lectura.toUpperCase();
            String cadenaRevertida = (new StringBuffer(cadenaMayuscula)).reverse().toString();
            line=cadenaRevertida;
            System.out.println("STATUS: Echo created: "+line);

            /* Enviar datagrama de respuesta */
            byte[] lineBytes = line.getBytes();
            DatagramPacket newPacket = new DatagramPacket(
                lineBytes, lineBytes.length, packet.getAddress(), packet.getPort());

            server.send(newPacket);
            System.out.println("STATUS: Echo sent");
            System.out.println("STATUS: Waiting for new echo");
        }
    }
}
```

## Cliente UDP

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

public class ClientUDP
{
    public static void main(String[] args) throws IOException
    {
        String serverName = "localhost"; // direccion local
        int serverPort = 54322;          // direccion del puerto

        DatagramSocket serviceSocket = null;
        InetAddress serverAddress = null;
        DatagramPacket recivePacket = null;

        try
        {
            /* Crear socket */
            serviceSocket = new DatagramSocket();
            serverAddress = InetAddress.getByName(serverName);
        }
        catch (SocketException e)
        {
            // Se ha importado socket exception
            System.err.println("Unknown: " + serverName);
            System.exit(1);
        }

        /* INICIALIZA ENTRADA POR TECLADO */
        BufferedReader stdIn = new BufferedReader( new InputStreamReader(System.in));

        System.out.println("Introduzca un texto a enviar (END para acabar)");
        String userInput;
        userInput = stdIn.readLine(); /* Cadena almacenada en "userInput" */

        /* Comprobar si el usuario quiere terminar servicio */
        while (userInput.compareTo("END") != 0)
        {
            /* Crear datagrama con la cadena escrito en el cuerpo */
            byte[] bytesToSend = userInput.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(bytesToSend, bytesToSend.length, serverAddress,
serverPort);

            try
            {
                /* crear socket */
                serviceSocket = new DatagramSocket();
                /* Enviar datagrama a traves del socket */
                serviceSocket.send(sendPacket);
            }
            catch (UnknownHostException e)
            {
                System.err.println("Unknown: " + serverName);
                System.exit(1);
            }
            catch (IOException iOException)
            {
            }

            System.out.println("STATUS: Waiting for the reply");

            /* Crear e inicializar un datagrama VACIO para recibir la respuesta */
            recivePacket = new DatagramPacket(new byte[bytesToSend.length], bytesToSend.length);

            /* Recibir datagrama de respuesta */
            serviceSocket.receive(recivePacket);

            /* Extraer contenido del cuerpo del datagrama en variable "line" */
            byte[] contenido = recivePacket.getData();
            String line = new String(contenido);
            System.out.println("echo: " + line);
            System.out.println("Introduzca un texto a enviar (END para acabar)");
            userInput = stdIn.readLine();
        }

        System.out.println("STATUS: Closing lient");

        // Cerrar socket cliente
        serviceSocket.close();
        System.out.println("STATUS: closed");
    }
}
```

## TAREA 3: Análisis

### Ejercicio 1:

Prueba local. Lance una instancia del servidor y otra del cliente y pruebe el correcto funcionamiento de la misma. Haga una captura de pantalla donde se vea la interacción de las dos instancias de la aplicación.

Para la realización de las pruebas de una forma sencilla y que puedan ver fácilmente hemos creado un script en windows para que se ejecuten los últimos archivos compilados de java (primeramente también lo probamos ejecutando dos aplicaciones Java a la vez en Eclipse como se mostraba en el campus virtual):

Primeramente lanzamos el servidor y después lanzamos el cliente:

### TCP

#### Servidor TCP:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redes_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>java ServerUDP
Waiting for a new UDP client
IP cliente: 127.0.0.1 Puerto cliente: 62668
STATUS: Echo created: TSET
STATUS: Echo sent
STATUS: Waiting for new echo
Waiting for a new UDP client
```

#### Cliente TCP:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redes_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>cmd /k
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redes_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>java ClientUDP
STATUS: El puerto (cliente) que se ha usado es: 59977
Introduzca un texto a enviar (END para acabar)
test
STATUS: Waiting for the reply
echo: TSET
Introduzca un texto a enviar (END para acabar)
END
STATUS: Closing client
STATUS: closed
```

## UDP

### Servidor UDP:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>java ServerUDP
Waiting for a new UDP client
IP cliente: 127.0.0.1 Puerto cliente: 62668
STATUS: Echo created: TSET
STATUS: Echo sent
STATUS: Waiting for new echo
Waiting for a new UDP client
```

### Cliente UDP:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>cmd /k
STATUS: El puerto (cliente) que se ha usado es: 59977
Introduzca un texto a enviar (END para acabar)
test
STATUS: Waiting for the reply
echo: TSET
Introduzca un texto a enviar (END para acabar)
END
STATUS: Closing client
STATUS: closed

C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>cmd /k
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_Servidor\Programas de la practica 2\Archivos compilados\UDP>|
```

Como podemos ver, tanto TCP como UDP hacen la función correctamente.

Por otra parte, hemos capturado tanto para TCP como UDP las tramas en Wireshark, y teniendo en cuenta los puertos que hemos seleccionado para cada programa (TCP Servidor 12345, UDP Servidor 54322) obtenemos las siguientes capturas de pantalla:

## TCP

Practica 2 - Captura TCP.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

tcp.port=12345

No.	Time	Source	Destination	Protocol	Length	Info
340	5.393109	127.0.0.1	127.0.0.1	TCP	56	58916 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
341	5.393144	127.0.0.1	127.0.0.1	TCP	56	12345 → 58916 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
342	5.393171	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
409	9.192080	127.0.0.1	127.0.0.1	TCP	58	58916 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=6
410	9.192105	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [ACK] Seq=1 Ack=7 Win=2161152 Len=0
411	9.192577	127.0.0.1	127.0.0.1	TCP	58	12345 → 58916 [PSH, ACK] Seq=1 Ack=7 Win=2161152 Len=6
412	9.192994	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=7 Ack=7 Win=2161152 Len=0
413	11.462783	127.0.0.1	127.0.0.1	TCP	49	58916 → 12345 [PSH, ACK] Seq=7 Ack=7 Win=2161152 Len=5
414	11.462809	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [ACK] Seq=7 Ack=12 Win=2161152 Len=0
415	11.463282	127.0.0.1	127.0.0.1	TCP	46	12345 → 58916 [PSH, ACK] Seq=7 Ack=12 Win=2161152 Len=2
416	11.463309	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=12 Ack=9 Win=2161152 Len=0
417	11.463902	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [FIN, ACK] Seq=9 Ack=12 Win=2161152 Len=0
418	11.463918	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [ACK] Seq=12 Ack=10 Win=2161152 Len=0
419	11.464702	127.0.0.1	127.0.0.1	TCP	44	58916 → 12345 [FIN, ACK] Seq=12 Ack=10 Win=2161152 Len=0
420	11.464733	127.0.0.1	127.0.0.1	TCP	44	12345 → 58916 [ACK] Seq=10 Ack=13 Win=2161152 Len=0

## UDP

Wireshark practica 2 UDP.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

udp.port=54322

No.	Time	Source	Destination	Protocol	Length	Info
290	12.418978	127.0.0.1	127.0.0.1	UDP	36	62668 → 54322 Len=4
291	12.430868	127.0.0.1	127.0.0.1	UDP	36	54322 → 62668 Len=4

Aquí podemos ver como por debajo TCP es más complejo que UDP, mandando todos los mensajes necesarios para que la comunicación entre los extremos sea fiable



## Ejercicio 2:

Varios Clientes. Lance una instancia del servidor y dos o más del cliente y pruebe el correcto funcionamiento de estos. Haga una captura de pantalla donde se vea la interacción en el servidor. Utiliza Wireshark para capturar las tramas enviadas y coméntalas.

Como ya tenemos los scripts anteriores solo tendríamos que, en vez de ejecutar solo un servidor y un cliente ejecutar dos clientes:

### Servidor:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redes_Cliente_Servidor\Programas de la practica 2\Archivos compilados\TCP>java ServerTCP
Waiting for a new TCP client
Connecting with: 127.0.0.1
port: 50356
Received from client pruebamuchosclientes
Sending to client SETNEILCSOHCUMABEURP
```

### Cliente 1:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Go Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP
Conexion localhost/127.0.0.1
STATUS: Conectado al servidor
STATUS: El puerto (cliente) que se ha usado es: 50356
Introduzca un texto a enviar (END para acabar)
pruebamuchosclientes
STATUS: Enviando pruebamuchosclientes
STATUS: Esperando eco
echo: SETNEILCSOHCUMABEURP
Introduzca un texto a enviar (END para acabar)
```

Para poder diferenciar los dos clientes hemos añadido a los programas que se muestre el puerto que genera el serviceSocket para el cliente (que lo asigna arbitrariamente) para así diferenciarlos:

```
Programas de la practica 2/ClientTCP.java
35 35 @@ -35,6 +35,7 @@ public static void main(String[] args) throws IOException
36 36 }
37 37 System.out.println("STATUS: Conectado al servidor ");
38 + System.out.println("STATUS: El puerto (cliente) que se ha usado es: " + serviceSocket.getLocalPort()); // Puerto local
38 39
39 40 /* Obtener texto por teclado */
40 41 String userInput;

Programas de la practica 2/ServerTCP.java
35 35 @@ -35,7 +35,7 @@ public static void main(String[] args) throws IOException
36 36 /* COMPLETAR Esperar conexiones entrantes */
37 37 client = server.accept();
38 - System.out.println("Connecting with: " + client.getInetAddress().getHostAddress());
38 + System.out.println("port: " + client.getPort());
38 + System.out.println("port: " + client.getPort()); // Puerto remoto (del cliente)
39 39
40 40 }
41 41 catch (IOException e)
```

Cliente2:

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Go
Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP
Conexion locallocalhost/127.0.0.1
STATUS: Conectado al servidor
STATUS: El puerto (cliente) que se ha usado es: 50545
Introduzca un texto a enviar (END para acabar)
segundocliente
STATUS: Enviando segundocliente
STATUS: Esperando eco
```

Como podemos ver, no se pueden conectar dos clientes al mismo socket del servidor, se queda el último en espera, porque el servidor ya está conectado con el primero.

Si quisiéramos que llegue el otro mensaje del cliente tendríamos que escribir END en el Cliente 1, para que así pueda llegar los mensajes del Cliente2:

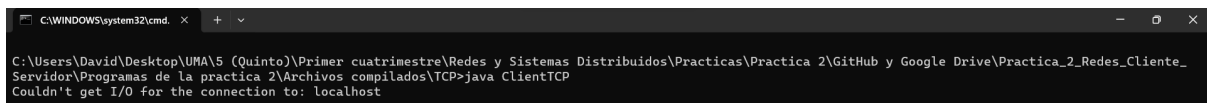
```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redres_Cliente_
Servidor\Programas de la practica 2\Archivos compilados\TCP>java ServerTCP
Waiting for a new TCP client
Connecting with: 127.0.0.1
port: 50356
Received from client pruebamuchosclientes
Sending to client SETNEILCSOHCUMABEURP
Received from client END
Closing connection with the client
Waiting for a new TCP client
Connecting with: 127.0.0.1
port: 50545
Received from client segundocliente
Sending to client ETNEILCODNUGES
```

### Ejercicio 3:

¿Pueden dos servidores en ejecución en la misma máquina escuchar en el mismo puerto? (Haz las pruebas y combinaciones que consideres necesarias para responder a estas preguntas. Justifica tu respuesta con capturas de pantalla o explicando qué pruebas has realizado y su resultado)

Al ejecutar dos servidores consecutivos, el segundo no se puede iniciar, ya que solo se puede crear un socket con el mismo puerto en un mismo dispositivo (que es como hemos probado en todos los ejercicios) aunque si se prueba desde dispositivos distintos si se podría, ya que la dirección IP sería distinta, y de esta forma los identificadores del socket serían distintos también:

### Servidor:



```
C:\WINDOWS\system32\cmd. x + v
C:\Users\David\Desktop\UMA\5 (Quinto)\Primer cuatrimestre\Redes y Sistemas Distribuidos\Practicas\Practica 2\GitHub y Google Drive\Practica_2_Redес_Cliente_Servidor\Programas de la practica 2\Archivos compilados\TCP>java ClientTCP
Couldn't get I/O for the connection to: localhost
```