

Programación de servicios y procesos

Tarea para PSP03.

Enunciado.

La tarea de la unidad está dividida en 2 actividades.

Actividad 3.1. El objetivo del ejercicio es crear una aplicación cliente/servidor que se comunique por el puerto 2000 y realice lo siguiente:

El servidor debe generar un número secreto de forma aleatoria entre el 0 al 100. El objetivo de cliente es solicitarle al usuario un número y enviarlo al servidor hasta que adivine el número secreto. Para ello, el servidor para cada número que le envía el cliente le indicará si es menor, mayor o es el número secreto del servidor.

Para resolver este ejercicio, he consultado los ejemplos que vienen en el temario, adecuándose su estructura muy bien a lo solicitado en la tarea.

Al tratarse de números secretos, he entendido que la seguridad en el envío de los datos era lo importante y por ello, he usado los ejemplos para conexiones TCP.

SERVIDOR.JAVA

```
/*
 * TAREA PSP03. EJERCICIO 1.
 * El objetivo del ejercicio es crear una aplicación cliente/servidor
que se
 * comunique por el puerto 2000 y realice lo siguiente:
 * El servidor debe generar un número secreto de forma aleatoria entre
el 0
 * al 100. El objetivo de cliente es solicitarle al usuario un número
y
 * enviarlo al servidor hasta que adivine el número secreto. Para
ello, el
 * servidor para cada número que le envía el cliente le indicará si es
menor,
 * mayor o es el número secreto del servidor.
 *
 * Esta clase genera la parte correspondiente al servidor
 * RECORDAR COMENTAR EL PACKAGE SI SE QUIERE COMPILAR FUERA DE
NETBEANS.
 */
package servidortcp;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 06/12/2020
 * @version 1
 */

public class ServidorTCP {
```

```

/**
 * El hecho de pedir números secretos, lo interpreto como parte de
una app
 * segura y por ello, hacemos las comunicaciones mediante TCP.
 * Sigo por ello, la estructura y ejemplo mostrados en el tema
para este tipo
 * de conexiones.
 * @param args the command line arguments
 * @param Puerto integer que indica el puerto de enlace
 * @param numeroCorrecto boolean inicializado en false para
tantear las
 * entradas desde el cliente.
 */

static final int Puerto = 2000;
boolean numeroCorrecto = false;

public ServidorTCP() {

    /**
     * @param numAleatorio objeto de la clase Random para
determinar el
     * numero secreto.
     * Lo mostramos en consola para verificar que recoge el dato.
     */
    int numAleatorio = (int) Math.round(Math.random() * 100);
    System.out.println("Numero secreto: " + numAleatorio);

    /**
     * try-catch para tratar la recogida y muestra de datos desde
el cliente.
     * la primera parte, hasta recibir la conexión con el cliente
sigue la
     * estructura del temario.
     * @throws Exception para mostrar mensaje de error en su caso.
     */

    try {
        // Inicio la escucha del servidor en un determinado puerto
        ServerSocket sSkServidor = new ServerSocket(Puerto);
        //Confirmamos que se recibe le puerto de escucha
        System.out.println("Escucho el puerto " + Puerto );
        // Espero a que se conecte un cliente y creo un nuevo
socket para
        el cliente
        Socket sSkCliente = sSkServidor.accept();
        //Confirmamos, por confianza, que se recibe.
        System.out.println("Servicio a cliente...");
        /**
         * Flujos abreviados (en temario se crean primero
variables
         * Input/OutputStream) de entrada y salida mediante
objetos
         * DataInputStream y DataOutputStream
         */
        DataInputStream flujo_entrada = new
DataInputStream(sSkCliente.getInputStream());
        DataOutputStream flujo_salida= new
DataOutputStream(sSkCliente.getOutputStream());
    }
    /**

```

```

        * Mientras reciba un dato incorrecto, se trata de forma
individual
        * la instrucción, dentro del bucle while
        * @param numCliente recibe el flujo de entrada con el
dato a tratar.
        */
        while(numeroCorrecto==false){
            int numCliente=flujo_entrada.readInt();
            /**
            * Mediante @if tratamos primero la posibilidad
correcta
            * y en el @else los posibles errores.
            * Utilizamos marcas de colorescapando en ANSI para
resaltar.
            */
            if(numCliente==numAleatorio){
                numeroCorrecto=true;
                flujo_salida.writeBoolean(numeroCorrecto);
                //Indicar por la salida que es la opción correcta
                flujo_salida.writeUTF("\033[36m" + numCliente + "
es el numero correcto. \n");
            }else{
                if(numCliente<numAleatorio){
                    numeroCorrecto=false;
                    flujo_salida.writeBoolean(numeroCorrecto);
                    //Indicar por la salida que es menor
                    flujo_salida.writeUTF("\033[33mEl numero "+
numCliente +" es menor que el requerido.\n");
                }else{
                    numeroCorrecto=false;
                    flujo_salida.writeBoolean(numeroCorrecto);
                    ////Indicar por la salida que es mayor
                    flujo_salida.writeUTF("\033[33mEl numero "+
numCliente +" es mayor que el requerido\n");
                }
            } //Fin if/else
        } //Fin while
        /**
        * Tras tratar los datos, se cierran conexiones y se avisa
de ello.
        */
        flujo_entrada.close();
        flujo_salida.close();
        System.out.println("Cerrando conexion.");
        sSkServidor.close();
        sSkCliente.close();
    } catch( Exception e ) {
        System.out.println( e.getMessage() );
    }
} //Fin constructor ServidorTCP

public static void main(String[] args) {
    // Iniciamos para poder recibir datos
    new ServidorTCP();
} //Fin main
} //Fin clase ServidorTCP

```

Podemos ejecutarlo en consola mediante **javac ServidorTCP.java** y luego **java ServidorTCP**.

CLIENTE.JAVA

```
/*
 * TAREA PSP03. EJERCICIO 1.
 * El objetivo del ejercicio es crear una aplicación cliente/servidor
que se
 * comunique por el puerto 2000 y realice lo siguiente:
 * El servidor debe generar un número secreto de forma aleatoria entre
el 0
 * al 100. El objetivo de cliente es solicitarle al usuario un número
y
 * enviarlo al servidor hasta que adivine el número secreto. Para
ello, el
 * servidor para cada número que le envía el cliente le indicará si es
menor,
 * mayor o es el número secreto del servidor.
 *
 * Esta clase genera la parte correspondiente al cliente
 * RECORDAR COMENTAR EL PACKAGE SI SE QUIERE COMPILAR FUERA DE
NETBEANS.
 */
package clientetcp;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
import java.util.Scanner;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 06/12/2020
 * @version 1
 */
public class ClienteTCP {

    /**
     * Al igual que con el servidor, aprovecho la mayor parte de la
estructura
     * y ejemplos mostrados en el tema para este tipo de conexiones.
     * @param args the command line arguments
     * @param Puerto integer que indica el puerto de enlace
     * @param HOST String que indica el canal de enlace. Al ser en un
equipo
     * local, lo haremos mediante localhost.
     */

    static final String HOST = "localhost";
    static final int Puerto = 2000;

    public ClienteTCP() {

        /**
         * @param numCliente integer para los datos que se introducen
a consola
         * para verificar.
         * @param numeroCorrecto boolean inicializado en false para
tantear las
         * entradas desde el cliente.
         * @param seguir boolean para manejar la entrada o no de datos
         * Usamos la clase Scanner para la recogida.
         */
    }
}
```

```

        int numCliente = 0;
        boolean numeroCorrecto = false;
        boolean seguir;
        Scanner teclado = new Scanner(System.in);

        /**
         * try-catch para tratar la recogida y muestra de datos desde
         * el cliente.
         * la primera parte, hasta recibir la conexión con el cliente
         * sigue la
         * estructura del temario.
         * Flujos abreviados (en temario se crean primero variables
         * Input/OutputStream) de entrada y salida mediante objetos
         * DataInputStream y DataOutputStream
         * @throws Exception para mostrar mensaje de error en su caso.
         */

        try{
            //Me conecto al servidor desde un determinado puerto
            Socket sSkCliente = new Socket( HOST , Puerto );
            DataOutputStream flujo_salida= new
DataOutputStream(sSkCliente.getOutputStream());
            DataInputStream flujo_entrada = new
DataInputStream(sSkCliente.getInputStream());
            /**
             * Mientras reciba un dato incorrecto, se trata de forma
             * individual
             * la instrucción, dentro del bucle while.
             */
            while(numeroCorrecto==false){
                do{
                    try{
                        /**
                         * try-catch para tratar errores de formato,
                         * por no usar
                         * enteros.
                         * @param texto String que recoge el dato para
                         * asegurar
                         * el correcto tratamiento de los errores.
                         * la primera parte, hasta recibir la conexión
                         * con el
                         * cliente sigue la estructura del temario.
                         * @throws NumberFormatException para mostrar
                         * mensaje
                         * de error en su caso.
                         */
                        seguir=false;
                        //Se pide un numero (0-100)
                        System.out.println("Debe introducir un numero
entre el 0 y el 100: ");
                        String texto=teclado.nextLine();
                        /**
                         * Lo parseamos para buscar errores a entero.
                         * Lo mostramos en consola para asegurar la
                         * entrada.
                         */
                        numCliente=Integer.parseInt(texto);
                        System.out.println("Ha introducido el " +
numCliente);

```

```

        /**
        * Mediante @if tratamos si el número es
        menor/mayor
        * al registrado desde el servidor.
        * Utilizamos marcas de colorescapando en ANSI
        para
        * resaltar la salida en consola.
        */
        if(numCliente < 0){
            System.out.println("\033[33mEl número
introducido es menor que 0");
            seguir=true;
        }
        if(numCliente > 100){
            System.out.println("\033[33mEl número
introducido es mayor que 100");
            seguir=true;
        }
    } catch (NumberFormatException ex){
        System.out.println("\033[31mNo ha introducido
un entero!");
        seguir=true;
    }
}while(seguir);
/**
* Verificado, Mandamos el dato al servidor, que nos
indica
* primero mediante un booleano si la opción era
correcta o no
* para seguir buscando, y segundo, el mensaje
correspondiente
* para mostrar por la consola.
*/
flujo_salida.writeInt(numCliente);
numeroCorrecto=flujo_entrada.readBoolean();
System.out.println(flujo_entrada.readUTF());
}
/**
* Tras tratar los datos, se cierran conexiones.
*/
teclado.close();
flujo_entrada.close();
flujo_salida.close();
sSkCliente.close();
} catch( Exception e ) {
    System.out.println( e.getMessage() );
}
} //Fin constructor ClienteTCP

/**
* @param args the command line arguments
*/
public static void main(String[] args) {
    new ClienteTCP();
}
}

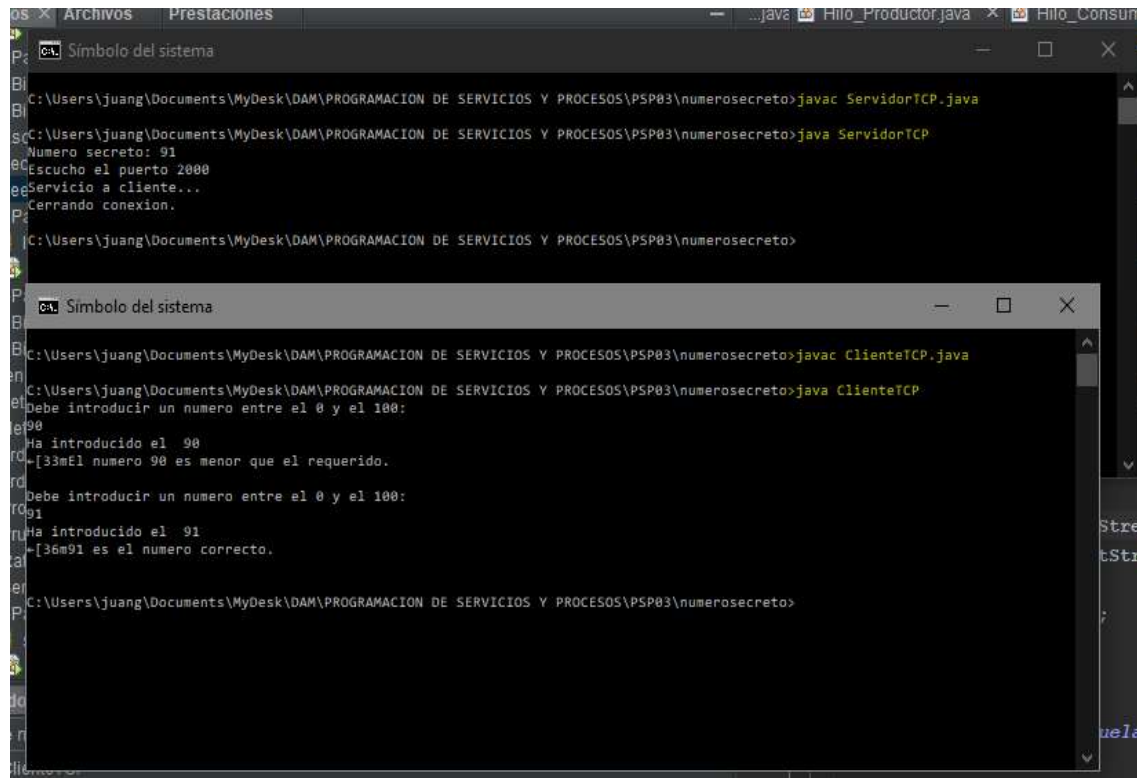
```

Podemos ejecutarlo en consola mediante **javac ClienteTCP.java** y luego **java ClienteTCP**.

Los he probado desde la Shell del sistema en Windows a través de dichos comandos y desde la propia consola de NetBeans.

Para la consola de Windows, hay que recordar que por el tipo de archivo (lo estoy aislando desde netbeans), es conveniente comentar el **package** para evitar problemas en la compilación.

El resultado se vería de la siguiente manera.

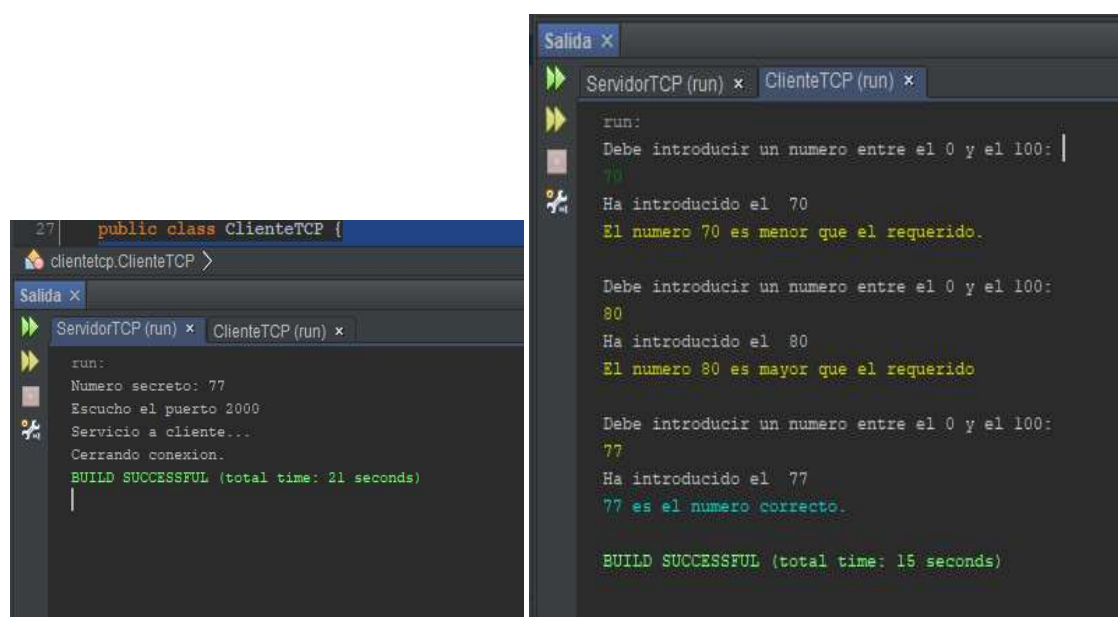


```
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\numerosecreto>javac ServidorTCP.java
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\numerosecreto>java ServidorTCP
Numero secreto: 91
Escucho el puerto 2000
Servicio a cliente...
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\numerosecreto>
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\numerosecreto>javac ClienteTCP.java
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\numerosecreto>java ClienteTCP
Debe introducir un numero entre el 0 y el 100:
90
Ha introducido el 90
[33mEl numero 90 es menor que el requerido.
Debe introducir un numero entre el 0 y el 100:
91
Ha introducido el 91
[36m91 es el numero correcto.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\numerosecreto>
```

En el caso de compilar desde Netbeans, se puede ver de esta forma:



```
27 public class ClienteTCP {
    clientetcp.ClienteTCP >

Salida x
ServidorTCP (run) x ClienteTCP (run) x
run:
Numero secreto: 77
Escucho el puerto 2000
Servicio a cliente...
Cerrando conexion.
BUILD SUCCESSFUL (total time: 21 seconds)

run:
Debe introducir un numero entre el 0 y el 100:
70
Ha introducido el 70
El numero 70 es menor que el requerido.
Debe introducir un numero entre el 0 y el 100:
80
Ha introducido el 80
El numero 80 es mayor que el requerido
Debe introducir un numero entre el 0 y el 100:
77
Ha introducido el 77
77 es el numero correcto.
BUILD SUCCESSFUL (total time: 15 seconds)
```

Actividad 3.2. El objetivo del ejercicio es crear una aplicación cliente/servidor que permita el envío de ficheros al cliente. Para ello, el cliente se conectará al servidor por el puerto 1500 y le solicitará el nombre de un fichero del servidor. Si el fichero existe, el servidor, le enviará el fichero al cliente y éste lo mostrará por pantalla. Si el fichero no existe, el servidor le enviará al cliente un mensaje de error. Una vez que el cliente ha mostrado el fichero se finalizará la conexión.

Como en la primera parte, he tomado los mismos ejemplos del temario, ya que parecen de igual importancia los resultados a obtener y he aplicado el mismo tipo de estructura, pero adaptando el ejercicio para que el cliente pueda seleccionar archivos añadidos al mismo directorio de la tarea.

He intentado documentar todo el código para una mejor comprensión sin tener que detallar o recortarlo en varias capturas que puedan entorpecer su lectura.

SERVIDORFILETCP.JAVA

```
/*
 * TAREA PSP03. EJERCICIO 2.
 * El objetivo del ejercicio es crear una aplicación cliente/servidor
 * que permita el envío de ficheros al cliente. Para ello, el cliente
se
 * conectará al servidor por el puerto 1500 y le solicitará el nombre
de
 * un fichero del servidor. Si el fichero existe, el servidor, le
enviará
 * el fichero al cliente y éste lo mostrará por pantalla. Si el
fichero no
 * existe, el servidor le enviará al cliente un mensaje de error. Una
vez
 * que el cliente ha mostrado el fichero se finalizará la conexión.
 *
 * Esta clase genera la parte correspondiente al servidor
 * RECORDAR COMENTAR EL PACKAGE SI SE QUIERE COMPILAR FUERA DE
NETBEANS.
 */
package servidorfiletcp;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 09/12/2020
 * @version 1
 */
public class ServidorFileTCP {

    /**
     * El hecho de pedir compartir documentos/archivos, lo interpreto
como
     * parte de una app segura y por ello,
     * hacemos las comunicaciones mediante TCP.

```



```

    * Sigo por ello, la estructura y ejemplo mostrados en el tema
para
    * este tipo de conexiones.
    * Las instrucciones piden conexión por puerto 1500, pedir nombre
del
    * fichero y ver si existe. Creamos las variables para controlarlo
    * @param args the command line arguments
    * @param Puerto integer que indica el puerto de enlace
    * @param nombreFichero String recoge el nombre facilitado
    * @param fileExist boolean inicializado en false para controlar
    * la existencia o no del fichero solicitado
    */

    static final int Puerto = 1500;
    String nombreFichero;
    boolean fileExist = false;

    public ServidorFileTCP( ) {

        /**
         * try-catch para tratar la recogida y muestra de datos desde
el cliente.
         * la primera parte, hasta recibir la conexión con el cliente
sigue la
         * estructura del temario.
         * @throws Exception para mostrar mensaje de error en su caso.
         */

        try {
            // Inicio la escucha del servidor en un determinado puerto
            ServerSocket sSkServidor = new ServerSocket(Puerto);
            //Confirmamos que se recibe le puerto de escucha
            System.out.println("Escucho el puerto " + Puerto );
            // Espero a que se conecte un cliente y creo un nuevo
socket para el cliente
            Socket sSkCliente = sSkServidor.accept();
            //Confirmamos, por confianza, que se recibe.
            System.out.println("Servicio a cliente...");
            /**
             * Flujos abreviados (en temario se crean primero
variables
             * Input/OutputStream) de entrada y salida mediante
objetos
             * DataInputStream y DataOutputStream
             */
            DataInputStream flujo_entrada = new
DataInputStream(sSkCliente.getInputStream());
            DataOutputStream flujo_salida= new
DataOutputStream(sSkCliente.getOutputStream());

            /**
             * Recibimos el nombre del fichero y por asegurar la
             * recogida, lo mostramos.
             */
            nombreFichero=flujo_entrada.readUTF();
            System.out.println("Fichero solicitado: " +
nombreFichero);
            /**
             * Creamos un objeto de la clase File para comprobar si
existe
             * correlación con nuestros archivos.

```

```

        * @see archivoPedido recibe el flujo de entrada para
asociarlo.
        * Mediante condicionales, comprobamos si existe o no.
        */
File archivoPedido = new File(nombreFichero);

if(!archivoPedido.exists()) {
    //No existe
    fileExist = false;
    flujo_salida.writeBoolean(fileExist);
    flujo_salida.writeUTF("No existen coincidencias con: "
+ nombreFichero);
} else {
    //Si existe:
    if(archivoPedido.isFile()){
        fileExist = true;
        flujo_salida.writeBoolean(fileExist);
        flujo_salida.writeUTF("Encontrada coincidencia
con: " + nombreFichero);
    }
    /**
     * try-catch para tratar posibles errores con el
archivo
     * @throws IOException para mostrar mensaje de
error en su caso.
     */
    try{
        /**
         * @see leeArchivo objeto de FileInputStream
para
         * leer el archivo recogido.
         */
        FileInputStream leeArchivo = new
FileInputStream(nombreFichero);
        /**
         * @param longArchivo integer para longitud
archivoPedido.
         */
        int longArchivo = (int)archivoPedido.length();

        flujo_salida.writeInt(longArchivo);
        /**
         * @param bytes array que recoge los bytes del
archivo
         * según su tamaño.
         */
        int bytes[] = new int[longArchivo];
        /**
         * @param final_archivo boolean inicializado
en false
         * para recorrer archivo hasta el final.
         * @param contador integer local para el
control
         * de la lectura del flujo de bytes.
         * @param bytesArchivo integer para la lectura
de bytes.
         * Leerá hasta llegar a -1, que nos marca el
punto final.
         */
        boolean final_archivo = false;
        int contador = 0;
        while(final_archivo == false){

```

```

        int bytesArchivo = leeArchivo.read();
        if(bytesArchivo != -1){
            bytes[contador] = bytesArchivo;
            flujo_salida.write(bytesArchivo);
        }else{
            final_archivo = true;
        }
        contador++;
    }
    //Cerramos el objeto InputSream
    leeArchivo.close();
} catch(IOException ex){
    System.out.println("Error en acceso a
archivo");
}
//Por asegurar que se ha llegado a este punto, lo
mostramos por pantalla
System.out.println("Enviando archivo: " +
nombreFichero);
/**
 * Nos queda controlar que hacer si hay coincidencias
 * en el nombre pero no es un archivo.
 */
} else{
    fileExist = false;
    flujo_salida.writeBoolean(fileExist);
    flujo_salida.writeUTF(nombreFichero + " no
corresponde con ningún archivo o ruta correcta.");
}
}
/**
 * Tras tratar los datos, se cierran conexiones y se avisa de
ello.
 */
flujo_entrada.close();
flujo_salida.close();
System.out.println("Cerrando conexion.");
sSkServidor.close();
sSkCliente.close();
} catch( Exception e ) {
    System.out.println( e.getMessage() );
}
}
} //Fin constructor ServidorFileTCP

public static void main(String[] args) {
    // TODO code application logic here
    new ServidorFileTCP();

} //Fin main
} //Fin clase ServidorFileTCP

```

Podemos ejecutarlo en consola mediante **javac ServidorFileTCP.java** y luego **java ServidorFileTCP**.

En ese caso, el servidor queda a la escucha, esperando la petición del cliente.

Este archivo sigue las mismas directrices que lo comentado anteriormente y aprovechando en parte los ejemplos de la plataforma.

CLIENTEFILETCP.JAVA

```
/*
 * TAREA PSP03. EJERCICIO 2.
 * El objetivo del ejercicio es crear una aplicación cliente/servidor
 * que permita el envío de ficheros al cliente. Para ello, el cliente
se
 * conectará al servidor por el puerto 1500 y le solicitará el nombre
de
 * un fichero del servidor. Si el fichero existe, el servidor, le
enviará
 * el fichero al cliente y éste lo mostrará por pantalla. Si el
fichero no
 * existe, el servidor le enviará al cliente un mensaje de error. Una
vez
 * que el cliente ha mostrado el fichero se finalizará la conexión.
 *
 * Esta clase genera la parte correspondiente al cliente
 * RECORDAR COMENTAR EL PACKAGE SI SE QUIERE COMPILAR FUERA DE
NETBEANS.
 */
package clienteftcp;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 09/12/2020
 * @version 1
 */
public class ClienteFileTCP {

    /**
     * Al igual que con el servidor, aprovecho la mayor parte de la
estructura
     * y ejemplo mostrados en el tema para este tipo de conexiones.
     * @param args the command line arguments
     * @param Puerto integer que indica el puerto de enlace
     * @param HOST String que indica el canal de enlace. Al ser en un
equipo
     * local, lo haremos mediante localhost.
     */

    static final String HOST = "localhost";
    static final int Puerto = 1500;

    public ClienteFileTCP(){
        /**
         * @param fileExist boolean para controlar
         * la existencia o no del fichero solicitado.
         */
        boolean fileExist;

        /**
         * try-catch para tratar la recogida y muestra de datos desde
el cliente.

```

```

    * la primera parte, hasta recibir la conexión con el cliente
sigue la
    * estructura del temario.
    * Flujos abreviados (en temario se crean primero variables
    * Input/OutputStream) de entrada y salida mediante objetos
    * DataInputStream y DataOutputStream
    * @throws Exception para mostrar mensaje de error en su caso.
    */
    try{
        //Me conecto al servidor desde un determinado puerto
        Socket sSkCliente = new Socket( HOST , Puerto );
        DataOutputStream flujo_salida= new
DataOutputStream(sSkCliente.getOutputStream());
        DataInputStream flujo_entrada = new
DataInputStream(sSkCliente.getInputStream());

        /**
         * @param teclado objeto clase Scanner para recogida de
         * la entrada de datos con ruta y/o nombre del fichero por
consola.
         * @param nombreFichero String recoge el nombre facilitado
para
         * pasarlo a continuación al flujo de salida.
         */

        Scanner teclado = new Scanner(System.in);
        System.out.println("Indique nombre del archivo: ");
        String nombreFichero = teclado.nextLine();

        /**
         * Lo mostramos para asegurar que se ha recogido antes de
mandarlo
         * al flujo de salida hacia el servidor.
         */
        System.out.println("El archivo buscado es: " +
nombreFichero);
        flujo_salida.writeUTF(nombreFichero);

        /**
         * Comprobamos si existe según el booleano que nos remita
el
         * servidor por el flujo de entrada y mostramos el mensaje
         * correspondiente.
         */
        fileExist = flujo_entrada.readBoolean();
        System.out.println(flujo_entrada.readUTF());

        //Si existe:
        if(fileExist == true){

            //Recogeremos el tamaño del archivo para crear un
array con ese tamaño
            /**
             * @param longArchivo integer para longitud/tamaño
archivo.
             * @param bytes array que recoge los bytes del
archivo
             * según su tamaño.
             */
            int longArchivo = flujo_entrada.readInt();
            int bytes[] = new int[longArchivo];

```

```

        /**
        * try-catch para tratar posibles errores con el
archivo
        * @throws IOException para mostrar mensaje de error
en su caso.
        */
        try{
            /**
            * @see copiaArchivo objeto de FileOutputStream
para
            * escribir el archivo copiado.
            */
            FileOutputStream copiaArchivo = new
FileOutputStream(nombreFichero + "(copia)");
            /**
            * Mediante un for recorreremos los bytes recogidos
desde el
            * servidor y se rellena el array
            */
            for(int i=0;i<bytes.length;i++){
                bytes[i]=flujo_entrada.read();
                copiaArchivo.write(bytes[i]);
            }
            //Cerramos el objeto OutputStream
            copiaArchivo.close();
        }catch(IOException ex){
            System.out.println("Error al crear archivo");
        }
    } //fin if/else control exists
} /**
* Tras tratar los datos, se cierran conexiones.
*/
flujo_entrada.close();
flujo_salida.close();
System.out.println("Cerrando conexion.");
teclado.close();
sSkCliente.close();
} catch( Exception e ) {
    System.out.println( e.getMessage() );
}
} //fin constructor ClienteFileTCP

public static void main(String[] args) {

    new ClienteFileTCP();
} //Fin main

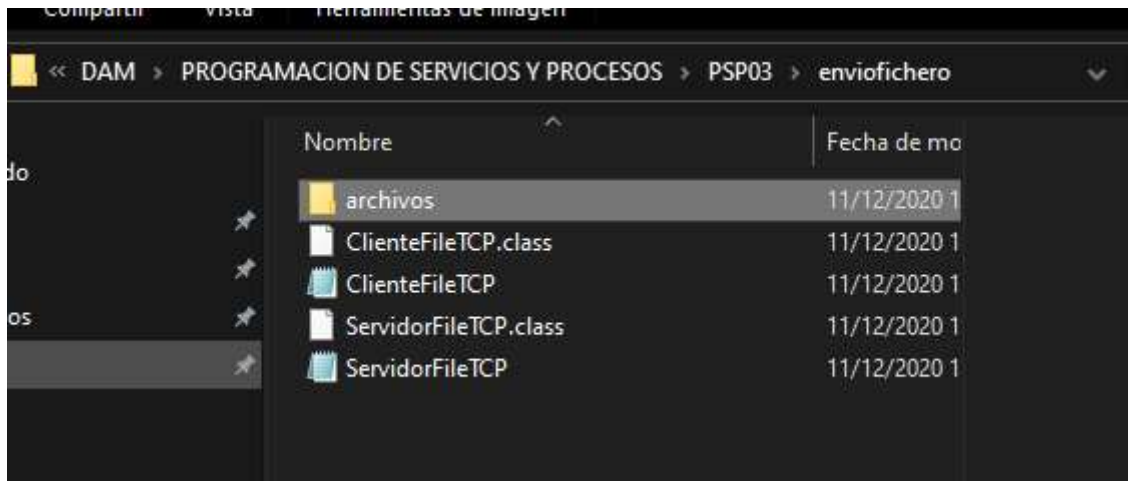
} //Fin ClienteFileTCP

```

Podemos ejecutarlo en consola mediante **javac ClienteFileTCP.java** y luego **java ClienteFileTCP**.

Tras esto podemos solicitar al servidor algún archivo, esperando que responda en su caso con un nuevo archivo, o notifique el error correspondiente.

Para poder copiar archivos, he creado en el mismo directorio una carpeta, denominada **archivos** con dos elementos con los que poder comprobar el funcionamiento correcto de la aplicación.



Podemos observar como si se introduce la ruta correcta, el servidor responde a la llamada con los datos correspondientes, y si no indica el error.

```

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>javac ServidorFileTCP.java

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>java ServidorFileTCP
Escucho el puerto 1500
Servicio a cliente...
Fichero solicitado: archivos/eventos-geek.jpg
Enviando archivo: archivos/eventos-geek.jpg
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>java ServidorFileTCP
Escucho el puerto 1500
Servicio a cliente...
Fichero solicitado: prueba.txt
Enviando archivo: prueba.txt
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>java ServidorFileTCP
Escucho el puerto 1500
Servicio a cliente...
Fichero solicitado: archivos/pruebaPSP03.txt
Enviando archivo: archivos/pruebaPSP03.txt
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>javac ClienteFileTCP.java

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>java ClienteFileTCP
Indique nombre del archivo:
archivos/eventos-geek.jpg
El archivo buscado es: archivos/eventos-geek.jpg
Encontrada coincidencia con: archivos/eventos-geek.jpg
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>java ClienteFileTCP
Indique nombre del archivo:
prueba.txt
El archivo buscado es: prueba.txt
No existen coincidencias con: prueba.txt
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>java ClienteFileTCP
Indique nombre del archivo:
archivos/pruebaPSP03.txt
El archivo buscado es: archivos/pruebaPSP03.txt
Encontrada coincidencia con: archivos/pruebaPSP03.txt
Cerrando conexion.

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\PSP03\enviofichero>

```

Tras el envío de datos, en caso de ser correcta la petición, los archivos son creados en el directorio:

