

Programación de servicios y procesos

Tarea para PSP04.

Enunciado.

La tarea de la unidad está dividida en 3 actividades.

Actividad 4.1. Modifica el ejercicio 1 de la unidad 3 para el servidor permita trabajar de forma concurrente con varios clientes.

Para resolver este ejercicio, he consultado los ejemplos que vienen en el temario, para poder incorporarlos al código con el que se resolvió la tarea del tema 3.

En todos los apartados de esta tarea, intentaré dejar documentado lo mejor posible el código para una mejor comprensión de cómo lo he resuelto.

Además de respetar la conexión TCP he aprovechado la estructura del ejemplo 2.3 del tema.

SERVIDORTHREAD.JAVA

```
/*
 * TAREA PSP04. EJERCICIO 1.
 * Proviene del tema 3. Se deben modificar los archivos de dicha tarea para
 * que el servidor permita trabajar de forma concurrente con varios clientes.
 *
 * 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 servidorthread;

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

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 14/01/2021
 * @version 1
 */
public class ServidorThread extends Thread{

    /**
     * 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 skCliente Socket de enlace entre procesos.
     * @param Puerto integer que indica el puerto de enlace
     * @param numeroCorrecto boolean inicializado en false para tantear las
     * entradas desde el cliente.
     */

    Socket skCliente;
    static final int Puerto = 2000;
    boolean numeroCorrecto = false;
```

```

/**
 * Inicializamos los valores que reciba el hilo
 * @param sCliente
 */

public ServidorThread(Socket sCliente){
    skCliente = sCliente;
}

public static void main(String[] args) {
    /**
     * 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 skServidor = 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
        while(true){
            // Se conecta un cliente
            Socket skCliente = skServidor.accept();
            System.out.println("Cliente conectado");

            // Atiendo al cliente mediante un thread
            new ServidorThread(skCliente).start();
        }
    } catch (Exception e) {
        System.out.println( e.getMessage() );
    }
} //Fin main

/**
 * @method run se encarga de realizar las tareas del hilo
 */
public void run(){
    /**
     * @param numAleatorio objeto de la clase Random para determinar el
     * número secreto.
     * Lo mostramos en consola para verificar que recoge el dato.
     */

    int numAleatorio = (int)Math.round(Math.random()*100);
    System.out.println("Número secreto: " + numAleatorio);

    try {
        /**
         * Flujos abreviados (en temario se crean primero variables
         * Input/OutputStream) de entrada y salida mediante objetos
         * DataInputStream y DataOutputStream
         */
        DataInputStream flujo_entrada = new
DataInputStream(skCliente.getInputStream());
        DataOutputStream flujo_salida= new
DataOutputStream(skCliente.getOutputStream());
        /**
         * @param numCliente integer para controlar la recepción
         * de dicho dato.
         */

        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 coloreo 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
número 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.
 */
System.out.println("Cerrando conexion.");
skCliente.close();
System.out.println("Cliente desconectado");
} catch( Exception e ) {
    System.out.println( e.getMessage() );
} //Fin bloque try-catch
} //Fin run()
} //Fin clase ServidorThread

```

CLIENTETHREAD.JAVA

```

/*
 * TAREA PSP04. EJERCICIO 1.
 * Proviene del tema 3. Se deben modificar los archivos de dicha tarea
para
 * que el servidor permita trabajar de forma concurrente con varios
clientes.
 *
 * 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 clientethread;

```

```

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

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 14/01/2021
 * @version 1
 */
public class ClienteThread {

    /**
     * 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=2000;

    public ClienteThread() {

        /**
         * @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;

        System.out.println("*****");
        System.out.println("ADIVINANDO UN NUMERO DEL 0 AL 100");
        System.out.println("*****");

        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 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 ClienteThread

public static void main(String[] args) {
    new ClienteThread();
} //Fin main
} //Fin clase ClienteThread

```

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

En la siguiente imagen se observa la conexión, emisión de números secretos según se conectan clientes y su respuesta.

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\juang> cd Documents\MyDesk\DAIM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea
PS C:\Users\juang> java ServidorThread
Escucho el puerto 2000
Cliente conectado
Numero secreto: 32
Cliente conectado
Numero secreto: 36
Cerrando conexion.
Cliente desconectado
Cerrando conexion.
Cliente desconectado

```

Para ello, ejecutamos los clientes en consola mediante **javac ClienteThread.java** y luego **java ClienteThread**.

En la primera imagen vemos un primer cliente y las interacciones con el servidor, introduciendo distintos valores numéricos o alfanuméricos para comprobar el correcto funcionamiento, así como el valor correcto y el cierre de la conexión.

```
Windows PowerShell
PS C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea> javac ClienteThread.java
PS C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea> java ClienteThread
*****
ADIVINANDO UN NUMERO DEL 0 AL 100
*****
Debe introducir un numero entre el 0 y el 100:
64
Ha introducido el 64
+{36m64 es el numero correcto.

PS C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea> java ClienteThread
*****
ADIVINANDO UN NUMERO DEL 0 AL 100
*****
Debe introducir un numero entre el 0 y el 100:
23
Ha introducido el 23
+{33mEl numero 23 es menor que el requerido.

Debe introducir un numero entre el 0 y el 100:
31
Ha introducido el 31
+{33mEl numero 31 es menor que el requerido.

Debe introducir un numero entre el 0 y el 100:
45
Ha introducido el 45
+{33mEl numero 45 es mayor que el requerido

Debe introducir un numero entre el 0 y el 100:
1
+{31mNo ha introducido un entero!
Debe introducir un numero entre el 0 y el 100:
32
Ha introducido el 32
+{36m32 es el numero correcto.

PS C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>
```

También probamos con un Cliente 2 para probar los distintos hilos mientras se mantiene la ejecución con el hilo primero, tal y como se observa en el servidor, que entrega números y respuestas según se comunican los distintos clientes.

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\juang> cd Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea
PS C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea> java ClienteThread
*****
ADIVINANDO UN NUMERO DEL 0 AL 100
*****
Debe introducir un numero entre el 0 y el 100:
25
Ha introducido el 25
+{33mEl numero 25 es menor que el requerido.

Debe introducir un numero entre el 0 y el 100:
36
Ha introducido el 36
+{36m36 es el numero correcto.

PS C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>
```

Actividad 4.2. Modifica el ejercicio 2 de la unidad 3 para el servidor permita trabajar de forma concurrente con varios clientes.

Como con el primer ejercicio, buena parte se basa en el código ya creado para resolver este parte de la tarea 3, manteniendo las conexiones TCP y añadiendo parte de las estructuras de los ejemplos mostrados para programación concurrente de la segunda parte del tema.

SERVIDORFILETHREAD.JAVA

```
/*
 * TAREA PSP04. EJERCICIO 2.
 * Proviene del tema 3. Se deben modificar los archivos de dicha tarea para
 * que el servidor permita trabajar de forma concurrente con varios clientes.
 *
 * 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 servidorfilethread;

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 14/01/2021
 * @version 1
 */

public class ServidorFileThread extends Thread{
    /**
     * 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 skCliente Socket de enlace entre procesos
     * @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
     */
    Socket skCliente;
    static final int Puerto = 1500;
    String nombreFichero;
    boolean fileExist=false;

    /**
     * Inicializamos los valores que reciba el hilo
     * @param sCliente
     */
    public ServidorFileThread(Socket sCliente ) {
        skCliente = sCliente;
    } //Fin ServidorFileTCP

    public static void main(String[] args) {
        /**
         * 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 skServidor = new ServerSocket(Puerto);
            //Confirmamos que se recibe le puerto de escucha
            System.out.println("Escucho el puerto " + Puerto );

            while(true){
                // Se conecta un cliente y genera su socket
                Socket skCliente = skServidor.accept();

```



```

        //Confirmamos, por confianza, que se recibe.
        System.out.println("Servicio conectado a cliente...");
    /**
        * Atiendo al cliente mediante un thread que se inicia con
start
        */
        new ServidorFileThread(skCliente).start();
    }
    } catch (Exception e) {
        System.out.println( e.getMessage() );
    } //Fin try-catch
} //Fin main

/**
 * @method run se encarga de realizar las tareas del hilo
 */
public void run() {
    try {
        /**
        * Flujos abreviados (en temario se crean primero variables
        * Input/OutputStream) de entrada y salida mediante objetos
        * DataInputStream y DataOutputStream
        */
        DataInputStream flujo_entrada = new
DataInputStream(skCliente.getInputStream());
        DataOutputStream flujo_salida= new
DataOutputStream(skCliente.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.
 */

    System.out.println("Cerrando conexion.");
    skCliente.close();
} catch (Exception e) {
    System.out.println( e.getMessage() );
}
} //Fin run()
} //Fin clase ServidorFileThread

```

CLIENTEFILETHREAD.JAVA

```
/*
 * TAREA PSP04. EJERCICIO 2.
 * Proviene del tema 3. Se deben modificar los archivos de dicha tarea para
 * que el servidor permita trabajar de forma concurrente con varios clientes.
 *
 * 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 clienteFileThread;

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 14/01/2021
 * @version 1
 */

public class ClienteFileThread {

    /**
     * Al igual que con el servidor, aprovecho la mayor parte de la estructura
     * y ejemplo mostrados en el tema para este tipo de conexiones.
     * El funcionamiento del cliente no se ve modificado, al ser el servidor
     * el que se encarga de procesar los hilos.
     * @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 ClienteFileThread(){
        /**
         * @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 skCliente = new Socket( HOST , Puerto );
        DataOutputStream flujo_salida= new
DataOutputStream(skCliente.getOutputStream());
        DataInputStream flujo_entrada = new
DataInputStream(skCliente.getInputStream());

        /**
         * @param teclado objeto clase Scanner para recogida de
         * la entrada de datos con ruta y/o nombre del fichero por
console.
         * @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/ruta 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-cath 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 OutputSream
                copiaArchivo.close();
            }catch(IOException ex){
                System.out.println("Error al crear archivo");
            }
        }
    } //fin if/else control exists

```

```

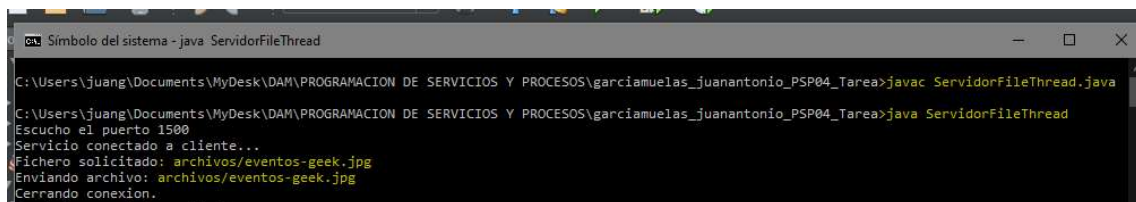
/**
 * Tras tratar los datos, se cierran conexiones.
 */
flujo_entrada.close();
flujo_salida.close();
System.out.println("Cerrando conexion.");
teclado.close();
skCliente.close();
} catch( Exception e ) {
    System.out.println( e.getMessage() );
}
} //fin constructor ClienteFileThread
public static void main(String[] args) {
    // TODO code application logic here
    new ClienteFileThread();
} //Fin main
} // Fin clase ClienteFileThread

```

Tras compilar podemos ejecutarlo en consola mediante `javac ServidorFileThread.java` y luego `java ServidorFileThread` y para el cliente `javac ClienteFileThread.java` y luego `java ClienteFileThread`.

Probamos con la conexión y petición a un cliente, donde responde y cierra. Observamos que la consola del servidor no retorna a la salida del directorio, por lo que se mantiene a la escucha de nuevos hilos.

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.

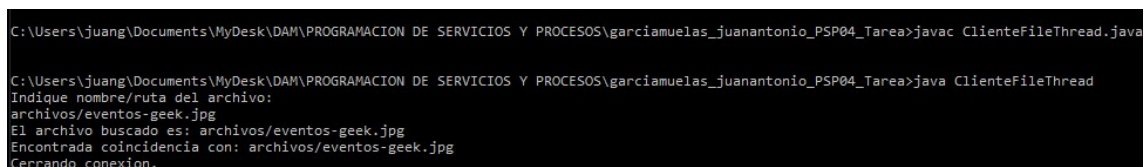


```

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ServidorFileThread.java
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ServidorFileThread
Escucho el puerto 1500
Servicio conectado a cliente...
Fichero solicitado: archivos/eventos-geek.jpg
Enviando archivo: archivos/eventos-geek.jpg
Cerrando conexion.

```

Vista desde el cliente.

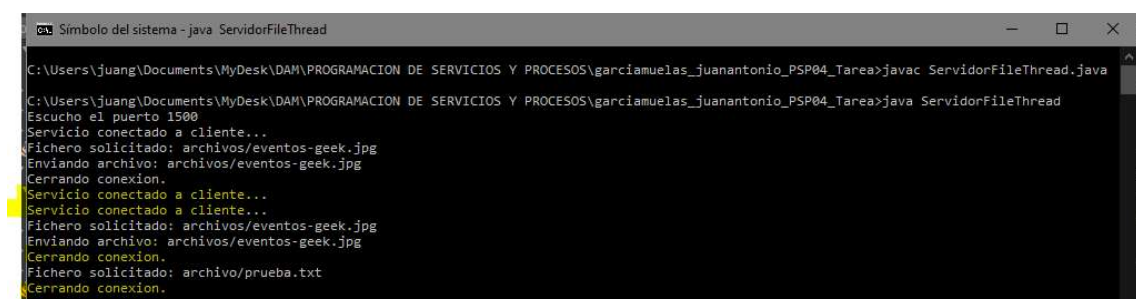


```

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ClienteFileThread.java
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ClienteFileThread
Indique nombre/ruta del archivo:
archivos/eventos-geek.jpg
El archivo buscado es: archivos/eventos-geek.jpg
Encontrada coincidencia con: archivos/eventos-geek.jpg
Cerrando conexion.

```

Probamos a interactuar con más de un hilo abriendo una nueva consola.



```

C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ServidorFileThread.java
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ServidorFileThread
Escucho el puerto 1500
Servicio conectado a cliente...
Fichero solicitado: archivos/eventos-geek.jpg
Enviando archivo: archivos/eventos-geek.jpg
Cerrando conexion.
Servicio conectado a cliente...
Servicio conectado a cliente...
Fichero solicitado: archivos/eventos-geek.jpg
Enviando archivo: archivos/eventos-geek.jpg
Cerrando conexion.
Fichero solicitado: archivo/prueba.txt
Cerrando conexion.

```

Probamos (tras borrar el archivo anterior) a hacer una petición correcta y otra con resultado negativo, respondiendo de forma eficaz.

```
Simbolo del sistema
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ClienteFileThread
Indique nombre/ruta del archivo:
archivo/prueba.txt
El archivo buscado es: archivo/prueba.txt
No existen coincidencias con: archivo/prueba.txt
Cerrando conexion.

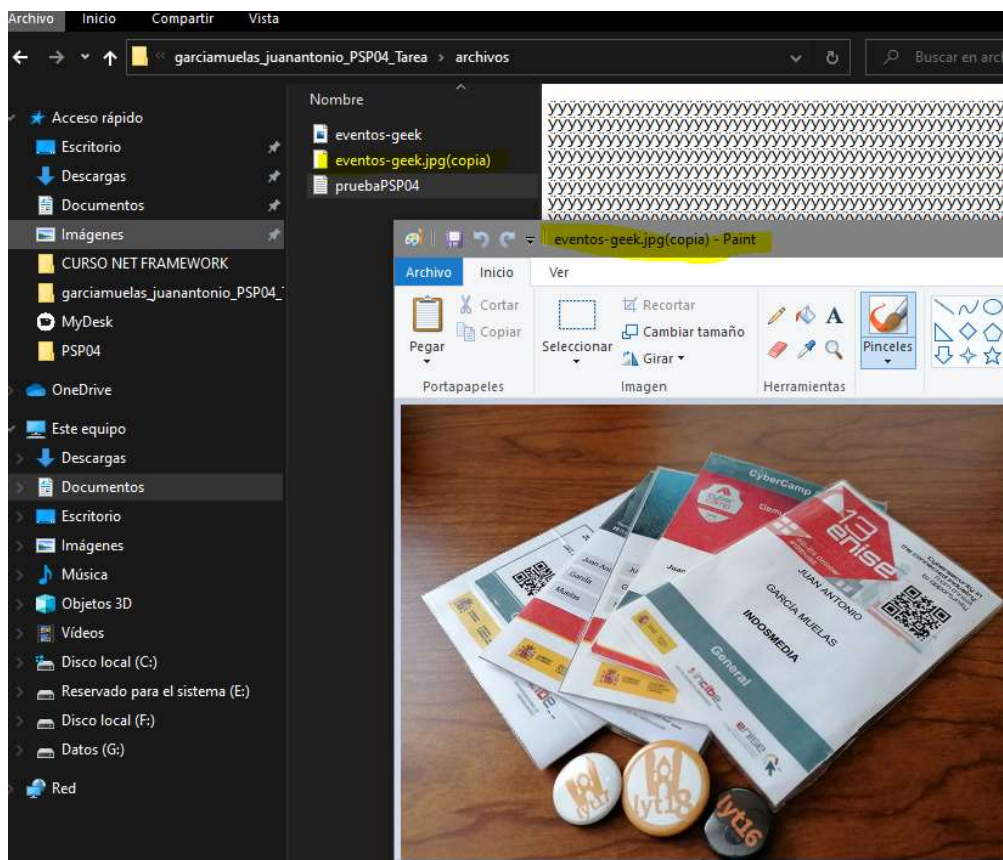
C:\Users\juang\Documents\MyDesk\DAM\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>

Simbolo del sistema
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\garciamuelas_juanantonio_PSP04_Tarea>java ClienteFileThread
Indique nombre/ruta 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\garciamuelas_juanantonio_PSP04_Tarea>
```

Comprobamos la inserción de la copia



Actividad 4.3. A partir del ejercicio anterior crea un servidor que una vez iniciada sesión a través de un nombre de usuario y contraseña específico (por ejemplo, javier / secreta) el sistema permita Ver el contenido del directorio actual, mostrar el contenido de un determinado archivo y salir.

Para realizar el ejercicio primero debes crear un diagrama de estados que muestre el funcionamiento del servidor.

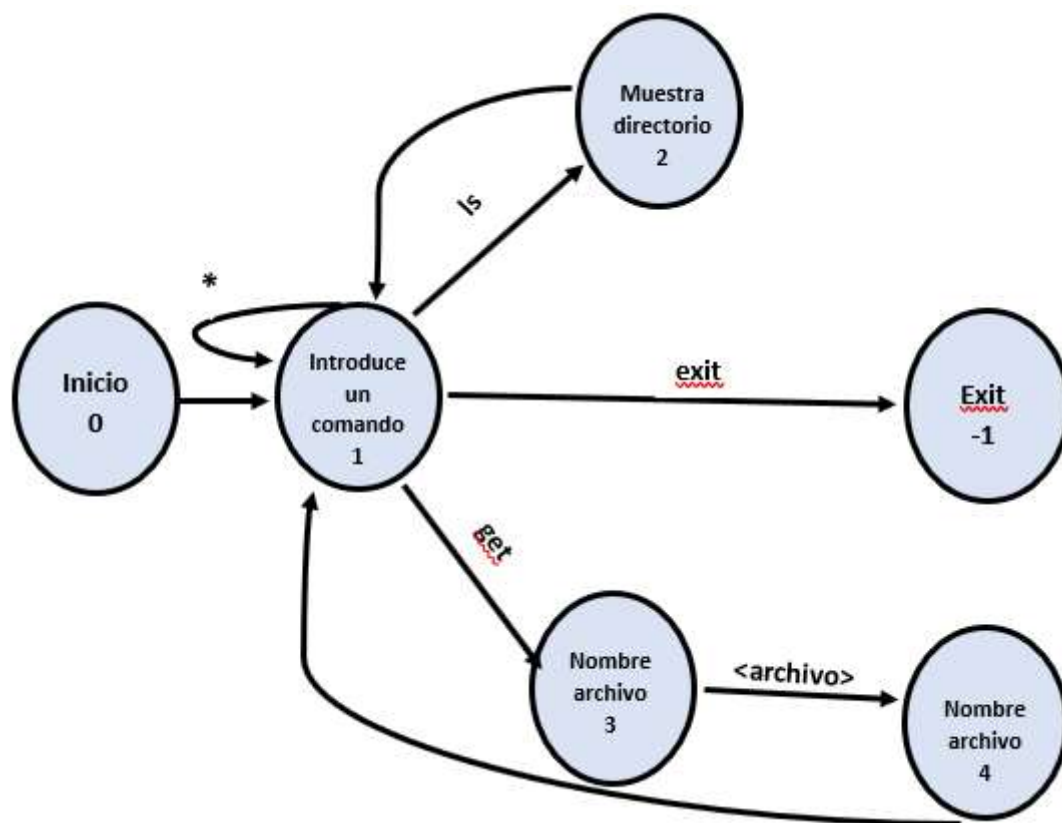
Como con los anteriores, buena parte se basa en el código ya creado para resolver este parte de la tarea 3, manteniendo las conexiones TCP y añadiendo parte de las estructuras vistas en los puntos 2.5 y 2.6, que orientan en gran manera la forma de ejecutar dicha tarea.

En base al punto 2.5, se desarrolla un diagrama de estados o autómata, para modelar los flujos de información y que se inicia en 0.

El cliente puede enviar los comandos:

- ls que va al estado 2 mostrando el contenido del directorio y vuelve automáticamente al estado 1.
- get que le lleva al estado 3 donde le solicita al cliente el nombre del archivo a mostrar. Al introducir el nombre del archivo se desplaza al estado 4 donde muestra el contenido del archivo y vuelve automáticamente al estado 1.
- exit que le lleva directamente al estado donde finaliza la conexión del cliente (estado -1).
- Cualquier otro comando hace que vuelva al estado 1 solicitándole al cliente que introduzca un comando válido.

El comportamiento puede observarse en el siguiente diagrama:



A continuación, se recoge el código correspondiente.

SERVIDORTHREADDIAGRAMA.JAVA

```
/*
 * TAREA PSP04. EJERCICIO 3.
 *
 * Proviene del tema 3. 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.
 *
 * Se deben modificar los archivos de dicha tarea para
```



```

* que el servidor permita trabajar de forma concurrente con varios clientes.
* A partir del ejercicio anterior crea un servidor que una vez iniciada
* sesión a través de un nombre de usuario y contraseña específico (en mi
* caso, juan / secreta) el sistema permita Ver el contenido del
* directorio actual, mostrar el contenido de un determinado archivo y salir.
* Para realizar el ejercicio primero debes crear un diagrama de estados
* que muestre el funcionamiento del servidor.
* Esta clase genera la parte correspondiente al cliente
* RECORDAR COMENTAR EL PACKAGE SI SE QUIERE COMPILAR FUERA DE NETBEANS.
*/

package servidorthreaddiagrama;

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 15/01/2021
 * @version 1
 */

public class ServidorThreadDiagrama extends Thread {

    /**
     * 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 skCliente Socket de enlace entre procesos
     * @param Puerto integer que indica el puerto de enlace
     * @param nombreFichero String recoge el nombre facilitado
     * @param usuario String recoge el nombre del usuario
     * @param contrasena String recoge su contraseña
     * @param fileExist boolean inicializado en false para controlar
     * la existencia o no del fichero solicitado
     */

    Socket skCliente;
    static final int Puerto = 1500;
    String nombreFichero, usuario, contrasena;
    boolean fileExist=false;

    /**
     * Inicializamos los valores que reciba el hilo
     * @param sCliente Socket de enlace entre procesos
     */

    public ServidorThreadDiagrama(Socket skCliente) {
        this.skCliente = skCliente;
        // TODO code application logic here
    }

    public static void main(String[] args) {
        /**
         * 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 skServidor = new ServerSocket(Puerto);
        //Confirmamos que se recibe le puerto de escucha
        System.out.println("Escucho el puerto " + Puerto );

        while(true){
            // Se conecta un cliente y genera su socket
            Socket skCliente = skServidor.accept();
            //Confirmamos, por confianza, que se recibe.
            System.out.println("Servicio conectado a cliente...");
            /**
             * Atiendo al cliente mediante un thread que se inicia con
start
             */
            new ServidorThreadDiagrama(skCliente).start();
        }
        catch (Exception e) {
            System.out.println( e.getMessage() );
        } //Fin try-catch
    } //Fin main

    /**
     * @method run se encarga de realizar las tareas del hilo
     */
    public void run(){
        try {

            /**
             * Flujos abreviados (en temario se crean primero variables
             * Input/OutputStream) de entrada y salida mediante objetos
             * DataInputStream y DataOutputStream
             */
            DataInputStream flujo_entrada = new
DataInputStream(skCliente.getInputStream());
            DataOutputStream flujo_salida= new
DataOutputStream(skCliente.getOutputStream());

            /**
             * Siguiendo el contenido del punto 2.5 y 2.6 modelamos el flujo
             * de información mediante un diagrama de estados/autómata,
             * lo hacemos a través de un entero.
             * @param estado integer que recoge ese dato (El servidor se
             * inicia a 0, para conectar con el cliente 1, listar con 2,
             * acceso archivos 3 y salir -1)
             * Lo manejamos a través de un while para poder asumir diferentes
             * hilos en el servidor.
             */

            int estado = 0;

            while ( estado!=1 ) {
                /**
                 * Recibimos el nombre del usuario y su contraseña.
                 * Al ser un ejemplo dado, no nos importa poder mostrar
                 * los datos para asegurar su recogida.
                 */
                usuario = flujo_entrada.readUTF();
                System.out.println("Nombre: " + usuario);
                contrasena = flujo_entrada.readUTF();
                System.out.println("Contraseña: " + contrasena);

                /**

```

```

        * Comprobamos datos y redirigimos según resultado.
        * Recordar que por personalización, uso de ejemplo juan.
        */
        if (usuario.equals("juan") && contrasena.equals("secreta")) {
            estado = 1;
            flujo_salida.writeUTF("Usuario correcto");
            flujo_salida.writeInt(estado);
        } else {
            flujo_salida.writeUTF("Usuario inválido");
            flujo_salida.writeInt(estado);
        } //Fin if-else de sesion
    } //Fin bucle while
    estado = 1;
    flujo_salida.writeInt(estado);

    /**
     * Recordar que los comandos son:
     * para el contenido del directorio actual "ls"
     * mostrar el contenido de un determinado archivo "get"
     * y salir "exit".
     *
     * Ahora que el usuario está autenticado, damos opciones.
     */

    while (estado != 0 ) {
        flujo_salida.writeUTF("Indique el comando a utilizar");
        /**
         * @param orden guarda los datos del comando solicitado
         * @param ls guarda los datos del directorio solicitado
         * @param carpeta objeto clase File que ayuda en la obtención
         * de archivos que comiencen con '.' y así evitamos algunos
         * errores que dependan de una extensión u otra.
         * @param longArchivo integer para longitud archivoPedido.
         */
        String orden;
        String[] ls;
        File carpeta = new File(".");
        int longArchivo;

        /**
         * Mediante @if @else ordenamos el flujo según la orden
         */
        orden = flujo_entrada.readUTF();
        if (orden.equals("ls")) {
            estado=2;
        } else {
            if (orden.equals("get")) {
                estado=3;
            } else {
                if (orden.equals("exit")) {
                    estado=-1;
                } else {
                    estado=1;
                }
            }
        }
    } //Fin opciones

    flujo_salida.writeInt(estado);
    /**
     * Con todo lo anterior podemos usar un bucle switch para
     * cada instrucción.
     */
    switch (estado) {
        case 2:
            System.out.println("Mostrar listado directorio");
            // Muestro el directorio
            ls = carpeta.list();
            longArchivo = ls.length;

```



```

del archivo
    /**
    * @param bytes array que recoge los bytes
    * según su tamaño.
    */
    int bytes[]=new int[longArchivo];
    /**
    * @param final_archivo boolean
    * para recorrer archivo hasta el final.
    * @param contador integer local para el
    * de la lectura del flujo de bytes.
    * @param bytesArchivo integer para la
    * Leerá hasta llegar a -1, que nos marca
    */
    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,
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.");
}
}
//Reseteamos el estado a 1 para que repita el while
estado = 1;
flujo_salida.writeInt(estado);
break;

case -1:
//Esta opción es para salir
System.out.println("\tEl cliente quiere salir");
estado =-1;
flujo_salida.writeInt(estado);
break;
} //Fin switch menú opciones
} //Fin while opciones

/**

```

```

        * Tras tratar los datos, se cierran conexiones y se avisa de
ello.
        */
        System.out.println("Cerrando conexion.");
        skCliente.close();
    } catch (Exception e) {
        System.out.println( e.getMessage() );
    }
} //Fin run()
} //Fin clase ServidorThreadDiagrama

```

CLIENTETHREADDIAGRAMA.JAVA

```

/*
 * TAREA PSP04. EJERCICIO 3.
 *
 * Proviene del tema 3. 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.
 *
 * Se deben modificar los archivos de dicha tarea para
 * que el servidor permita trabajar de forma concurrente con varios clientes.
 * A partir del ejercicio anterior crea un servidor que una vez iniciada
 * sesión a través de un nombre de usuario y contraseña específico (por
 * ejemplo, javier / secreta) el sistema permita Ver el contenido del
 * directorio actual, mostrar el contenido de un determinado archivo y salir.
 * Para realizar el ejercicio primero debes crear un diagrama de estados
 * que muestre el funcionamiento del servidor.
 * Esta clase genera la parte correspondiente al cliente
 * RECORDAR COMENTAR EL PACKAGE SI SE QUIERE COMPILAR FUERA DE NETBEANS.
 */

package clientethreaddiagrama;

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 15/01/2021
 * @version 1
 */

public class ClienteThreadDiagrama {

    /**
     * Al igual que con el servidor, aprovecho la mayor parte de la estructura
     * y ejemplo mostrados en el tema para este tipo de conexiones.
     * El funcionamiento del cliente no se ve modificado, al ser el servidor
     * el que se encarga de procesar los hilos.
     * @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;
    String usuario, contrasena;

```

```

public ClienteThreadDiagrama() {
    /**
     * @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 skCliente = new Socket( HOST , Puerto );
        DataOutputStream flujo_salida= new
DataOutputStream(skCliente.getOutputStream());
        DataInputStream flujo_entrada = new
DataInputStream(skCliente.getInputStream());

        /**
         * @param estado integer que recoge ese dato (El servidor se
         * inicia a 0, para conectar con el cliente 1, listar con 2,
         * acceso archivos 3 y salir -1)
         * @param orden guarda los datos del comando solicitado.
         * @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.
         */

        int estado =0;
        String orden;
        int longArchivo=0;
        Scanner teclado = new Scanner(System.in);

        //Solicitamos usuario y contraseña
        while(estado!=1){
            System.out.println("Introduzca usuario: ");
            usuario =teclado.nextLine();
            System.out.println("Introduzca contraseña: ");
            contrasena=teclado.nextLine();
            //Enviamos el nombre de usuario y contraseña mediante
flujo_salida
            flujo_salida.writeUTF(usuario);
            flujo_salida.writeUTF(contrasena);
            System.out.println(flujo_entrada.readUTF());
            estado=flujo_entrada.readInt();
        }
        estado=flujo_entrada.readInt();
        //Una vez comprobados correctamente usuario y contraseña, se nos
da a elegir lo que quiere hacer
        while(estado!=-1){
            System.out.println(flujo_entrada.readUTF());
            orden=teclado.nextLine();
            flujo_salida.writeUTF(orden);
            estado=flujo_entrada.readInt();
            //Se nos devuelve el comando transformado en estado(String a
Int) para entrar en el Switch
            switch(estado){
                case 2:
                    //Para mostrar el contenido del directorio (ls)

```

```

        longArchivo = flujo_entrada.readInt();
        if(longArchivo==0){
            //Si la ubicación está vacía
            System.out.println(flujo_entrada.readUTF());
        }else{
            System.out.println("Contenido del directorio
actual: ");

            for(int i=0;i<longArchivo;i++){
                System.out.println(flujo_entrada.readUTF());
            }
        }
        estado=flujo_entrada.readInt();
        break;

    case 3:
        System.out.println("Indique nombre/ruta 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 bytes[] = new int[longArchivo];

            /**
             * try-cath 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

estado=flujo_entrada.readInt();
break;
case -1:
    //Este es el caso en el que se desea salir
    estado=flujo_entrada.readInt();
    break;

} //Fin switch
} //Fin while

/**
 * Tras tratar los datos, se cierran conexiones.
 */
flujo_entrada.close();
flujo_salida.close();
System.out.println("Cerrando conexion.");
teclado.close();
skCliente.close();
} catch (Exception e) {
    System.out.println( e.getMessage() );
}
} //Fin constructor ClienteThreadDiagrama

public static void main(String[] args) {
    // TODO code application logic here
    new ClienteThreadDiagrama();
} //Fin main
} //Fin clase ClienteThreadDiagrama

```

La primera parte, consiste en comprobar si accedemos bien por la autenticación.

Tras el primer rechazo, accedemos con las credenciales correctas y nos muestra la petición.

The image shows two screenshots of a Java IDE. The top window, titled 'Simbolo del sistema - java.ServidorThreadDiagrama', shows the output of the server program. It starts with 'Escucho el puerto 1500', then 'Servicio conectado a cliente...', followed by 'Nombre: Pedro', 'Contraseña: 1234', 'Nombre: Juan', and 'Contraseña: secreta'. The bottom window, titled 'Simbolo del sistema', shows the output of the client program. It starts with 'Introduzca usuario:', followed by 'Pedro', 'Introduzca contraseña:', followed by '1234', 'Usuario inválido', 'Introduzca usuario:', followed by 'Juan', 'Introduzca contraseña:', followed by 'secreta', 'Usuario correcto', and finally 'Indique el comando a utilizar'.

```

C:\Users\juang\Documents\MyDesk\DAW\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ServidorThreadDiagrama.java
C:\Users\juang\Documents\MyDesk\DAW\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ServidorThreadDiagrama
Escucho el puerto 1500
Servicio conectado a cliente...
Nombre: Pedro
Contraseña: 1234
Nombre: Juan
Contraseña: secreta

C:\Users\juang\Documents\MyDesk\DAW\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ClientethreadDiagrama.java
C:\Users\juang\Documents\MyDesk\DAW\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ClientethreadDiagrama
Introduzca usuario:
Pedro
Introduzca contraseña:
1234
Usuario inválido
Introduzca usuario:
Juan
Introduzca contraseña:
secreta
Usuario correcto
Indique el comando a utilizar

```

Conectamos un segundo cliente y realizamos distintas peticiones para ver las respuestas, retornando tras ello al estado de espera.


```
Simbolo del sistema - java ServidorThreadDiagrama
C:\Users\juang\Documents\MyDesk\DA\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ServidorThreadDiagrama.java
C:\Users\juang\Documents\MyDesk\DA\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ServidorThreadDiagrama
Escucho el puerto 1500
Servicio conectado a cliente...
Nombre: Pedro
Contraseña: 1234
Nombre: Juan
Contraseña: secreta
Mostrar listado directorio
Servicio conectado a cliente...
Nombre: Juan
Contraseña: secreta
Fichero solicitado: archivos/pruebaPSP04
Fichero solicitado: archivos/copia.txt

Simbolo del sistema
C:\Users\juang\Documents\MyDesk\DA\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>javac ClienteThreadDiagrama.java
C:\Users\juang\Documents\MyDesk\DA\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ClienteThreadDiagrama
Introduzca usuario:
Pedro
Introduzca contraseña:
1234
Usuario inválido
Introduzca usuario:
Juan
Introduzca contraseña:
secreta
Usuario correcto
Indique el comando a utilizar
get
Contenido del directorio actual:
archivos
ClienteFileThread.class
ClienteFileThread.java
ClienteThread.class
ClienteThread.java
ClienteThreadDiagrama.class
ClienteThreadDiagrama.java
ServidorFileThread.class
ServidorThread.class
ServidorThreadDiagrama.class
ServidorThreadDiagrama.java
Indique nombre/ruta del archivo:
archivos/copia.txt
El archivo buscado es: archivos/copia.txt
No existen coincidencias con: archivos/copia.txt
Indique el comando a utilizar
```

Tras las pruebas, cerramos de forma correcta la conexión con el servidor.

```
Simbolo del sistema
C:\Users\juang\Documents\MyDesk\DA\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>java ClienteThreadDiagrama
Introduzca usuario:
Juan
Introduzca contraseña:
secreta
Usuario correcto
Indique el comando a utilizar
get
Indique nombre/ruta del archivo:
archivos/copia.txt
El archivo buscado es: archivos/copia.txt
No existen coincidencias con: archivos/copia.txt
Indique el comando a utilizar
exit
Cerrando conexión.
C:\Users\juang\Documents\MyDesk\DA\PROGRAMACION DE SERVICIOS Y PROCESOS\garciamuelas_juanantonio_PSP04_Tarea>
```