

Tarea PSP03



Yanet López Rodríguez
19/01/2024

Indice

• Portada	pg. 1
• Indice	pg. 2
• Introducción	pg. 3
• Ejercicio 1	pg. 4 - 8
◦ Manual de usuario	pg. 4 - 7
▪ Clase Hilo	pg. 4 - 5
▪ Clase Servidor	pg. 6
▪ Clase Cliente	pg. 6 - 7
◦ Pruebas	pg. 8
• Ejercicio 2	pg. 9 - 13
◦ Manual de usuario	pg. 9 - 12
▪ Clase Hilo	pg. 9 - 10
▪ Clase Servidor	pg. 11
▪ Clase Cliente	pg. 11 - 12
◦ Pruebas	pg. 13
• Conclusiones	pg. 14
• Recursos	pg. 14
• Bibliografía	pg. 14

Introducción

En esta actividad deberemos realizar las siguientes dos 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.

♥ 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.

En este archivo haremos un manual en el que explicaremos cada código y los probaremos.



♥ Actividad 3.1.

El objetivo del ejercicio es crear una aplicación cliente/servidor que se comuniquen 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 realizar esta actividad crearemos tres clases “Hilo”, “Servidor” y “Cliente”. Primero crearemos la clase “Hilo”, en ella debemos crear un nuevo objeto “Socket” y crear el constructor.

```
public class Hilo extends Thread{  
    //CREAMOS UN NUEVO OBJETO Socket  
    private Socket socket;  
  
    //CREAMOS EL CONSTRUCTOR DE LA CLASE  
    public Hilo(Socket socket){  
        this.socket = socket;  
    }  
}
```

Ahora creamos el método “run” y en él mostramos por pantalla (en el servidor) un mensaje para verificar que los clientes inician correctamente, creamos la instancia para leer los datos del “Socket”, creamos la instancia para escribir los datos en él y creamos un “try catch”.

Dentro del “try” especificamos el “Socket” para leer su flujo de entrada y salida, creamos un nuevo entero para almacenar en él el número recibido y lo inicializamos. Instanciamos la clase “Random” y creamos el entero “numeroSecreto” asignándole un número aleatorio generado por la instancia, y por último mostramos en el servidor el número secreto y al cliente el mensaje "Adivina el número secreto."

```
//CREAMOS EL MÉTODO RUN  
public void run() {  
    //MOSTRAMOS POR PANTALLA EN EL SERVIDOR UN MENSAJE PARA VERIFICAR QUE LOS CLIENTES INICIAN CORRECTAMENTE  
    System.out.println("El cliente se ha conectado correctamente");  
  
    //CREAMOS LA INSTANCIA PARA LEER LOS DATOS DEL SOCKET  
    DataInputStream inputStream = null;  
    //CREAMOS LA INSTANCIA PARA ESCRIBIR LOS DATOS EN EL SOCKET  
    DataOutputStream outputStream = null;  
  
    try {  
        //ESPECIFICAMOS EL SOCKET PARA LEER SU FLUJO DE ENTRADA  
        inputStream = new DataInputStream(socket.getInputStream());  
        //ESPECIFICAMOS EL SOCKET PARA LEER SU FLUJO DE SALIDA  
        outputStream = new DataOutputStream(socket.getOutputStream());  
  
        //CREAMOS UN NUEVO ENTERO PARA ALMACENAR EN EL EL NÚMERO RECIBIDO Y LO INICIALIZAMOS  
        int numeroRecibido = 0;  
  
        //INSTANCIAMOS LA CLASE RANDOM  
        Random random = new Random();  
        //CREAMOS EL ENTERO numeroSecreto ASIGNÁNDOLE UN NÚMERO ALEATORIO GENERADO POR LA INSTANCIA  
        int numeroSecreto = random.nextInt(101);  
  
        //MOSTRAMOS EN EL SERVIDOR EL NÚMERO SECRETO  
        System.out.println("Numero secreto: " + numeroSecreto);  
  
        //MOSTRAMOS AL CLIENTE EL MENSAJE "Adivina el número secreto."  
        outputStream.writeUTF("Adivina el número secreto.");  
    }  
}
```

Dentro del try también crearemos un bucle “do while” el cual se ejecutará mientras el número recibido sea distinto al secreto.

Dentro del bucle “do while” guardamos en su variable el número recibido, y lo mostramos en el servidor, si el número recibido es igual al número secreto mostramos "Has ganado!!!" al cliente, si el número recibido es mayor al número secreto mostramos "El número secreto es menor" y si el número recibido es menor al número secreto mostramos "El número secreto es mayor".

Ahora escribimos en el flujo de salida un booleano que nos dirá si el número secreto y el recibido son iguales o no, cerramos la conexión y lo comunicamos en el servidor y por último cerramos los flujos de entrada y salida de los datos.

En resumen, esta clase se encarga de la comunicación con un cliente que se conecta a un servidor, permitiendo al cliente adivinar un número secreto.

```
//CREAMOS UN BUCLE do while EL CUAL SE EJECUTARÁ MIENTRAS EL NÚMERO RECIBIDO SEA DISTINTO AL SECRETO
do{
    //GUARDAMOS EN SU VARIABLE EL NÚMERO RECIBIDO
    numeroRecibido = inputStream.readInt();

    //MOSTRAMOS EN EL SERVIDOR EL NÚMERO RECIBIDO
    System.out.println("Número recibido: " + numeroRecibido);

    //SI EL NÚMERO RECIBIDO ES IGUAL AL NÚMERO SECRETO MOSTRAMOS "Has ganado!!!" AL CLIENTE
    if(numeroRecibido == numeroSecreto){
        ouputStream.writeUTF("Has ganado!!!");
    //SI EL NÚMERO RECIBIDO ES MAYOR AL NÚMERO SECRETO MOSTRAMOS "El número secreto es menor"
    } else if(numeroRecibido > numeroSecreto){
        ouputStream.writeUTF("El número secreto es menor\n");
    //SI EL NÚMERO RECIBIDO ES MENOR AL NÚMERO SECRETO MOSTRAMOS "El número secreto es mayor"
    } else{
        ouputStream.writeUTF("El número secreto es mayor\n");
    }

    //ESCRIBIMOS EN EL FLUJO DE SALIDA UN BOOLEANO QUE NOS DIRÁ SI EL NÚMERO SECRETO Y EL RECIBIDO SON IGUALES O NO
    ouputStream.writeBoolean(numeroRecibido == numeroSecreto);
}while(numeroRecibido != numeroSecreto);

//CERRAMOS LA CONEXIÓN Y LO COMUNICAMOS EN EL SERIDOR
socket.close();
System.out.println("Cliente desconectado");

//SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
} catch (IOException ex) {
    Logger.getLogger(Hilo.class.getName()).log(Level.SEVERE, null, ex);
} finally {
    try {
        //CERRAMOS LOS FLUJOS DE ENTRADA Y SALIDA DE LOS DATOS
        inputStream.close();
        ouputStream.close();

        //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
    } catch (IOException ex) {
        Logger.getLogger(Hilo.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```


La segunda clase que crearemos será la de “*Servidor*”, en ella crearemos un “*main*” con un “*try catch*”. Dentro de este “*try*” creamos un “*ServerSocket*” que esperará conexiones del puerto 2000, mostramos por pantalla un mensaje para que se pueda verificar que el servidor esta iniciado correctamente y creamos un bucle “*while*” para que el servidor esté siempre a la espera de nuevas conexiones.

Dentro del bucle “*while*” aceptamos las conexiones entrantes y se la asignamos a “*socket*”, creamos un nuevo objeto de la clase “*Hilo*” y lo inicializamos pasándole el “*Socket*” de la conexión como parámetro y por último ejecutamos el objeto “*Hilo*” creado. Si se produce una excepción el “*catch*”, este lo captura, mostrando el error y el “*stack trace*”.

En resumen, crea un servidor que espera conexiones en el puerto 2000. Cuando una conexión entrante llega, crea un objeto de la clase Hilo y lo inicializa pasándole el socket de la conexión como parámetro. Luego, ejecuta el objeto Hilo. El servidor está diseñado para estar siempre a la espera de nuevas conexiones a través de un bucle infinito.

```
public class Servidor {
    public static void main(String[] args){
        try {
            //CREAMOS UN ServerSocket QUE ESPERARÁ CONEXIONES DEL PUERTO 2000
            ServerSocket servidor = new ServerSocket(2000);
            //MOSTRAMOS POR PANTALLA UN MENSAJE PARA QUE SE PUEDA VERIFICAR QUE EL SERVIDOR ESTA INICIADO CORRECTAMENTE
            System.out.println("El servidor ha iniciado correctamente.");

            //CREAMOS UN BUCLE PARA QUE EL SERVIDOR ESTÉ SIEMPRE A LA ESPERA DE NUEVAS CONEXIONES
            while(true){
                //ACEPTAMOS LAS CONEXIONES ENTRANTES Y SE LA ASIGNAMOS A socket
                Socket socket = servidor.accept();
                //CREAMOS UN NUEVO OBJETO DE LA CLASE Hilo Y LO INICIALIZAMOS PASÁNDOLE EL SOCKET DE LA CONEXIÓN COMO PARÁMETRO
                Hilo hilo = new Hilo(socket);
                //EJECUTAMOS EL OBJETO Hilo CREADO
                hilo.start();
            }
            //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
        } catch (IOException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Por último creamos la clase “*cliente*”, en ella crearemos un “*main*” con un “*try catch*”. Dentro de este “*try*” creamos un nuevo “*Socket*” y lo conectamos al servidor el cual se encuentra en “*localhost*” en el puerto 2000, creamos un nuevo “*DataInputStream*” para leer datos del “*Socket*”, un nuevo “*DataOutputStream*” para escribir datos del “*Socket*”, creamos una variable llamada “*salir*” y la inicializamos en “*false*” y por último mostramos “*Adivina el número secreto.*” al cliente.

```
public class Cliente {
    public static void main(String[] args){
        try {
            //CREAMOS UN NUEVO Socket Y LO CONECTAMOS AL SERVIDOR EL CUAL SE ENCUENTRA EN "localhost" EN EL PUERTO 2000
            Socket socket = new Socket("localhost", 2000);

            //CREAMOS UN NUEVO DataInputStream PARA LEER DATOS DEL Socket
            DataInputStream inputStream = new DataInputStream(socket.getInputStream());
            //CREAMOS UN NUEVO DataOutputStream PARA ESCRIBIR DATOS DEL Socket
            DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());

            //CREAMOS UNA VARIABLE LLAMADA salir Y LA INICIALIZAMOS EN false
            boolean salir = false;

            //MOSTRAMOS Adivina el número secreto. POR PANTALLA
            System.out.println(inputStream.readUTF());
        }
    }
}
```

Ahora creamos un bucle “while” el cual se ejecutará mientras “salir” sea “false”. Dentro del bucle leemos el número ingresado por consola y lo escribimos en el “Socket”.

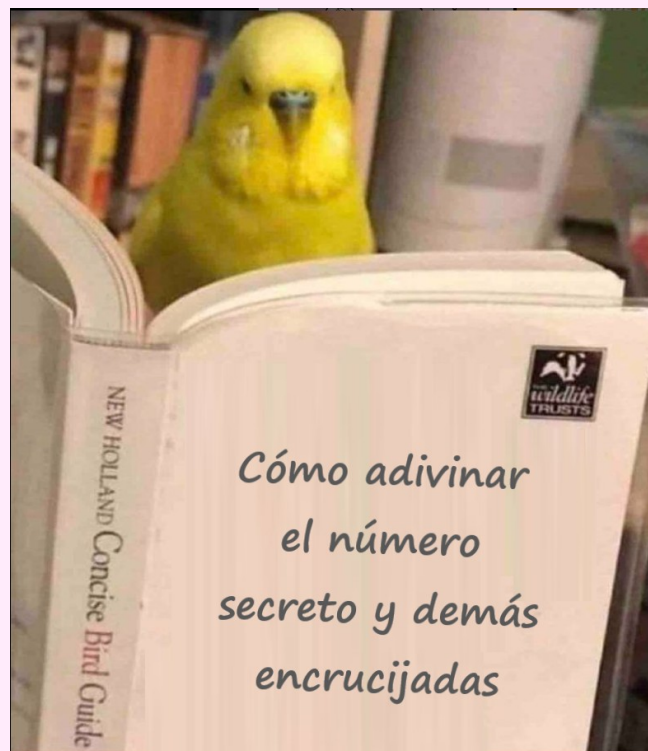
Mostramos por pantalla si el número secreto es mayor, menor o igual que el recibido, igualamos la variable “salir” a un valor booleano del “Socket” que será false hasta que el número secreto sea igual al número recibido y por último cerramos la conexión.

Si se produce una excepción el “catch”, este lo captura lo muestra junto a el “stack trace”.

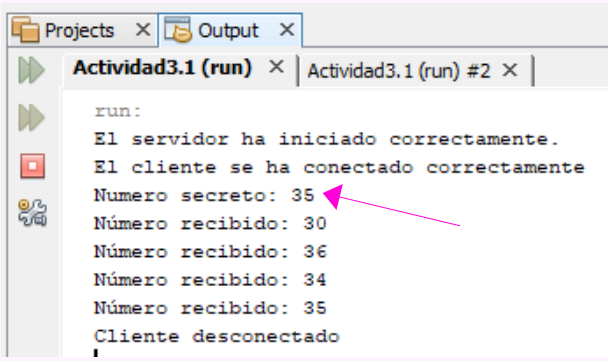
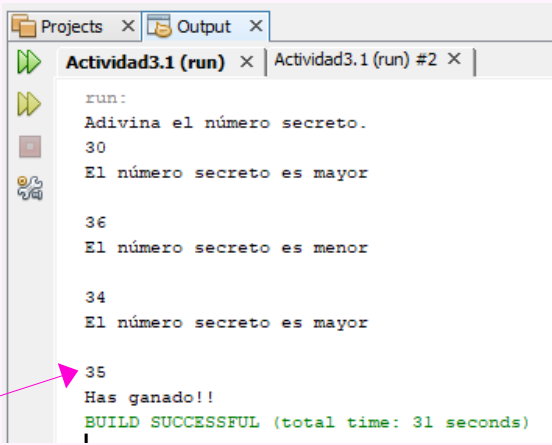
```
//CREAMOS UN BUCLE do while EL CUAL SE EJECUTARÁ MIENTRAS salir SEA false
do {
    //LEEMOS EL NÚMERO INGRESADO POR CONSOLA Y LO ESCRIBIMOS EN EL Socket
    Scanner scn = new Scanner(System.in);
    outputStream.writeInt(scn.nextInt());
    //MOSTRAMOS POR PANTALLA SI EL NÚMERO SECRETO ES MAYOR, MENOR O IGUAL QUE EL RECIBIDO
    System.out.println(inputStream.readUTF());
    //IGUALAMOS LA VARIABLE SALIR A UN VALOR BOOLEANO DEL Socket QUE SERÁ false HASTA QUE
    //EL NÚMERO SECRETO SEA IGUAL AL NÚMERO RECIBIDO
    salir = inputStream.readBoolean();
}while(!salir);

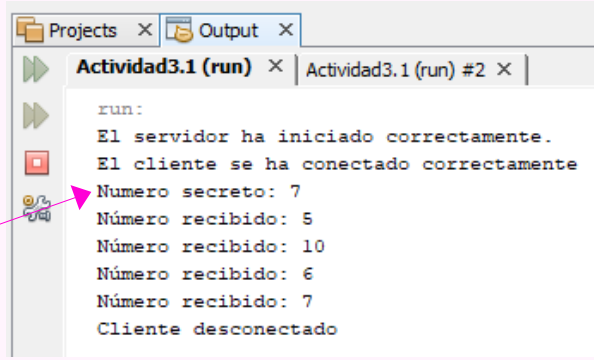
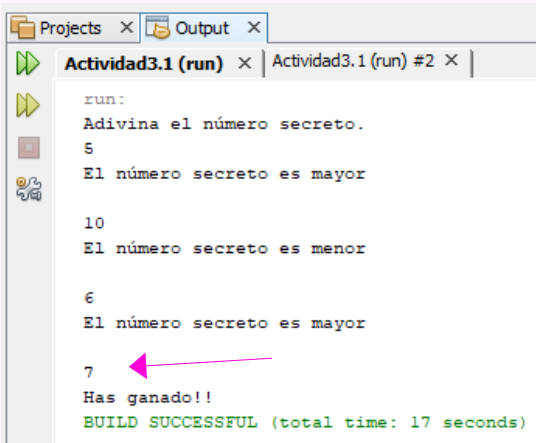
//CERRAMOS LA CONEXIÓN
socket.close();

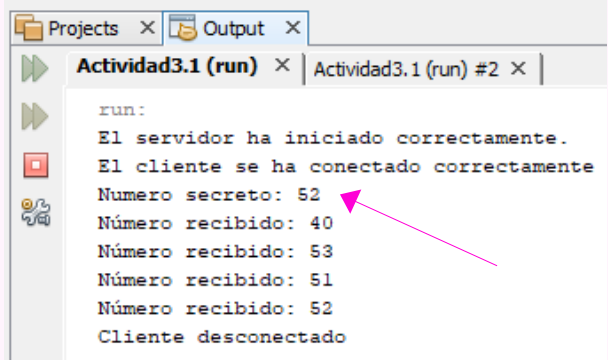
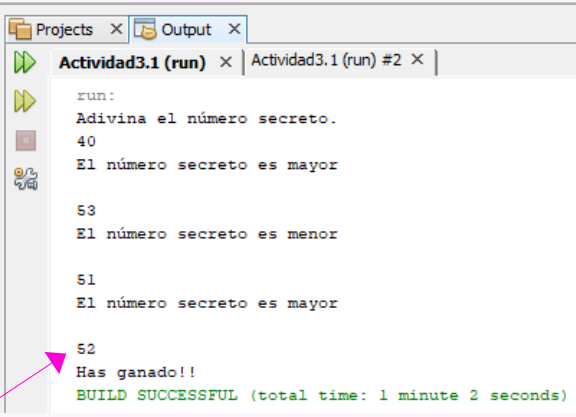
//SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
}catch(IOException ex){
    Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
}
```



★ Pruebas

Servidor	Cliente
 <pre> run: El servidor ha iniciado correctamente. El cliente se ha conectado correctamente Numero secreto: 35 Número recibido: 30 Número recibido: 36 Número recibido: 34 Número recibido: 35 Cliente desconectado </pre>	 <pre> run: Adivina el número secreto. 30 El número secreto es mayor 36 El número secreto es menor 34 El número secreto es mayor 35 Has ganado!! BUILD SUCCESSFUL (total time: 31 seconds) </pre>

Servidor	Cliente
 <pre> run: El servidor ha iniciado correctamente. El cliente se ha conectado correctamente Numero secreto: 7 Número recibido: 5 Número recibido: 10 Número recibido: 6 Número recibido: 7 Cliente desconectado </pre>	 <pre> run: Adivina el número secreto. 5 El número secreto es mayor 10 El número secreto es menor 6 El número secreto es mayor 7 Has ganado!! BUILD SUCCESSFUL (total time: 17 seconds) </pre>

Servidor	Cliente
 <pre> run: El servidor ha iniciado correctamente. El cliente se ha conectado correctamente Numero secreto: 52 Número recibido: 40 Número recibido: 53 Número recibido: 51 Número recibido: 52 Cliente desconectado </pre>	 <pre> run: Adivina el número secreto. 40 El número secreto es mayor 53 El número secreto es menor 51 El número secreto es mayor 52 Has ganado!! BUILD SUCCESSFUL (total time: 1 minute 2 seconds) </pre>

♥ 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.

Para realizar esta actividad crearemos tres clases “Hilo”, “Servidor” y “Cliente”. Primero crearemos la clase “Hilo”, en ella debemos crear un nuevo objeto “Socket” y crear el constructor.

```
public class Hilo extends Thread{
    //CREAMOS UN NUEVO OBJETO Socket
    private Socket socket;

    //CREAMOS EL CONSTRUCTOR DE LA CLASE
    public Hilo(Socket socket){
        this.socket = socket;
    }
}
```

Ahora creamos el método “run” y mostramos por pantalla (en el servidor) un mensaje para verificar que los clientes inician correctamente, creamos la instancia para leer y escribir los datos del “socket”. y creamos un “try catch”. Dentro del “try” especificamos el “Socket” para leer su flujo de entrada y salida, leemos la ruta y la guardamos en su variable y creamos un objeto “File” usando la variable “ruta”.

```
//CREAMOS EL MÉTODO RUN
@Override
public void run(){
    //MOSTRAMOS POR PANTALLA EN EL SERVIDOR UN MENSAJE PARA VERIFICAR QUE LOS CLIENTES INICIAN CORRECTAMENTE
    System.out.println("El cliente se ha conectado correctamente");

    //CREAMOS LA INSTANCIA PARA LEER LOS DATOS DEL SOCKET
    DataInputStream inputStream = null;
    //CREAMOS LA INSTANCIA PARA ESCRIBIR LOS DATOS EN EL SOCKET
    DataOutputStream outputStream = null;

    try {
        //ESPECIFICAMOS EL SOCKET PARA LEER SU FLUJO DE ENTRADA
        inputStream = new DataInputStream(socket.getInputStream());
        //ESPECIFICAMOS EL SOCKET PARA LEER SU FLUJO DE SALIDA
        outputStream = new DataOutputStream(socket.getOutputStream());

        //LEEMOS LA RUTA Y LA GUARDAMOS EN SU VARIABLE
        String ruta = inputStream.readUTF();
        //CREAMOS UN OBJETO File USANDO LA VARIABLE ruta
        File file = new File(ruta);
    }
}
```



¿Sabías que "run" significa correr en inglés?

Creamos un “if” para comprobar si la ruta existe, si existe escribimos en el flujo de salida que la ruta si existe, creamos un objeto “BufferedReader” que leerá el contenido de la ruta usando “FileReader” para abrir el archivo, creamos dos nuevos Strings y los inicializamos, creamos un bucle “while” que lee las líneas de “reader” hasta que no hallan más, añadiendo la línea leída a “contenido” junto con el caracter de retorno (\r) y una nueva línea (\n).

Cerramos objeto “BufferedReader” utilizando el método “close()”, convertimos el contenido del archivo en bytes, escribimos en el flujo de salida el tamaño del contenido, recorremos cada elemento de “contenidoArchivo” escribiendo en el flujo de salida el valor de cada uno y por último cerramos la conexión y lo comunicamos en el Servidor.

Si la ruta no existe lo escribimos en el flujo de salida.

Por último cerramos los flujos de entrada y salida de los datos. Si se produce una excepción el “catch”, este lo captura lo muestra junto a el “stack trace”.

En resumen, este código implementa un hilo de ejecución que maneja la conexión con un cliente en un servidor, verifica si se envía una ruta válida, lee el contenido de un archivo de esa ruta y lo envía de vuelta al cliente a través del socket.

```
//SI LA RUTA EXISTE EJECUTAMOS EL CÓDIGO
if(file.exists()){
    //ESCRIBIMOS EN EL FLUJO DE SALIDA QUE LA RUTA SI EXISTE
    ouputStream.writeBoolean(true);

    //CREAMOS UN OBJETO BufferedReader QUE LEERÁ EL CONTENIDO DE LA RUTA USANDO FileReader PARA ABRIR EL ARCHIVO
    BufferedReader reader = new BufferedReader(new FileReader(ruta));
    //CREAMOS DOS NUEVOS Strings Y LOS INICIALIZAMOS
    String linea, contenido = "";

    //CREAMOS UN BUCLE while QUE LEE LAS LINEAS DE reader HASTA QUE NO HALLAN MÁS
    while((linea = reader.readLine()) != null){
        //LE AÑADIMOS LA LINEA LEÍDA A contenido JUNTO CON EL CARACTER DE RETORNO \r Y UNA NUEVA LÍNEA \n
        contenido += linea + "\r\n";
    }

    //CERRAMOS OBJETO BufferedReader UTILIZANDO EL MÉTODO "close()"
    reader.close();

    //CONVERTIMOS EL CONTENIDO DEL ARCHIVO EN BYTES
    byte[] contenidoArchivo = contenido.getBytes();
    //ESCRIBIMOS EN EL FLUJO DE SALIDA EL TAMAÑO DEL CONTENIDO
    ouputStream.writeInt(contenidoArchivo.length);

    //RECORREMOS CADA ELEMENTO DE contenidoArchivo
    for (int i = 0; i < contenidoArchivo.length; i++) {
        //Y ESCRIBIMOS EN EL FLUJO DE SALIDA EL VALOR DE CADA UNO
        ouputStream.writeByte(contenidoArchivo[i]);
    }

    //CERRAMOS LA CONEXIÓN Y LO COMUNICAMOS EN EL SERVIDOR
    socket.close();
    System.out.println("Cliente desconectado");
}

//SI LA RUTA NO EXISTE LO ESCRIBIMOS EN EL FLUJO DE SALIDA
}else{
    ouputStream.writeBoolean(false);
}

//SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
} catch (IOException ex) {
    Logger.getLogger(Hilo.class.getName()).log(Level.SEVERE, null, ex);
} finally {
    try {
        //CERRAMOS LOS FLUJOS DE ENTRADA Y SALIDA DE LOS DATOS
        inputStream.close();
        ouputStream.close();
    } catch (IOException ex) {
        Logger.getLogger(Hilo.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

La segunda clase que crearemos será la de “*Servidor*”, en ella crearemos un “*main*” con un “*try catch*”. Dentro de este “*try*” creamos un “*ServerSocket*” que esperará conexiones del puerto 1500, mostramos por pantalla un mensaje para que se pueda verificar que el servidor está iniciado correctamente y creamos un bucle “*while*” para que el servidor esté siempre a la espera de nuevas conexiones.

Dentro del bucle “*while*” aceptamos las conexiones entrantes y se la asignamos a “*socket*”, creamos un nuevo objeto de la clase “*Hilo*” y lo inicializamos pasándole el “*Socket*” de la conexión como parámetro y por último ejecutamos el objeto “*Hilo*” creado. Si se produce una excepción el “*catch*”, este lo captura, mostrando el error y el “*stack trace*”.

En resumen, crea un servidor que espera conexiones en el puerto 2000. Cuando una conexión entrante llega, crea un objeto de la clase *Hilo* y lo inicializa pasándole el *socket* de la conexión como parámetro. Luego, ejecuta el objeto *Hilo*. El servidor está diseñado para estar siempre a la espera de nuevas conexiones a través de un bucle infinito.

```
public class Servidor {
    public static void main(String[] args){
        try {
            //CREAMOS UN ServerSocket QUE ESPERARÁ CONEXIONES DEL PUERTO 1500
            ServerSocket servidor = new ServerSocket(1500);
            //MOSTRAMOS POR PANTALLA UN MENSAJE PARA QUE SE PUEDA VERIFICAR QUE EL SERVIDOR ESTÁ INICIADO CORRECTAMENTE
            System.out.println("El servidor ha iniciado correctamente.");

            //CREAMOS UN BUCLE PARA QUE EL SERVIDOR ESTÉ SIEMPRE A LA ESPERA DE NUEVAS CONEXIONES
            while(true){
                //ACEPTAMOS LAS CONEXIONES ENTRANTES Y SE LA ASIGNAMOS A socket
                Socket socket = servidor.accept();

                //CREAMOS UN NUEVO OBJETO DE LA CLASE Hilo Y LO INICIALIZAMOS PASÁNDOLE EL SOCKET DE LA CONEXIÓN COMO PARÁMETRO
                Hilo hilo = new Hilo(socket);
                //EJECUTAMOS EL OBJETO Hilo CREADO
                hilo.start();
            }
        } catch (IOException ex) {
            //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Por último creamos la clase “*Cliente*”, en ella crearemos un “*main*” con un “*try catch*”. Dentro de este “*try*” creamos un nuevo “*Socket*” y lo conectamos al servidor el cual se encuentra en “*localhost*” en el puerto 1500, creamos un nuevo “*DataInputStream*” para leer datos del “*Socket*”, un nuevo “*DataOutputStream*” para escribir datos del “*Socket*”, le pedimos al cliente la ruta del archivo, especificamos el delimitador y guardamos la ruta en su variable y escribimos la ruta en el flujo de salida.

```
public class Cliente {
    public static void main(String[] args){
        try {
            //CREAMOS UN NUEVO Socket Y LO CONECTAMOS AL SERVIDOR EL CUAL SE ENCUENTRA EN "localhost" EN EL PUERTO 1500
            Socket socket = new Socket("localhost", 1500);

            //CREAMOS UN NUEVO DataInputStream PARA LEER DATOS DEL Socket
            DataInputStream inputStream = new DataInputStream(socket.getInputStream());
            //CREAMOS UN NUEVO DataOutputStream PARA ESCRIBIR DATOS DEL Socket
            DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());

            //LE PEDIMOS AL CLIENTE LA RUTA DEL ARCHIVO EJ: C:\Users\yanet\Desktop\xd.txt
            System.out.println("Introduce la ruta del archivo: ");
            Scanner scn = new Scanner(System.in);
            //ESPECIFICAMOS EL DELIMITADOR Y GUARDAMOS LA RUTA EN SU VARIABLE
            scn.useDelimiter("\n");
            String ruta = scn.next();

            //ESCRIBIMOS LA RUTA EN EL FLUJO DE SALIDA
            outputStream.writeUTF(ruta);
        }
    }
}
```

Creamos un “if” para comprobar si la ruta existe, si existe leemos la longitud del contenido del archivo, creamos la variable “*contenido*” con la misma longitud del contenido, recorremos la variable asignándole a cada byte el valor leído de “*inputStream*” y le mostramos al cliente el contenido del archivo. Si la ruta del archivo no es correcta mostramos “El archivo no existe” al cliente. Para finalizar cerramos la conexión. Si se produce una excepción el “catch”, este lo captura lo muestra junto a el “stack trace”.

En resumen, este código representa un cliente que se conecta a un servidor a través de un socket. El cliente solicita al usuario la ruta de un archivo y la envía al servidor a través del socket. Si la ruta existe, el servidor lee el contenido del archivo y lo envía de vuelta al cliente, quien lo muestra por pantalla. Si la ruta no existe, el servidor envía un mensaje al cliente indicando que el archivo no existe. Finalmente, se cierra la conexión con el servidor.

```
//SI LA RUTA EXISTE
if (inputStream.readBoolean()){
    //LEEMOS LA LONGITUD DEL CONTENIDO DEL ARCHIVO
    int longitud = inputStream.readInt();
    //CREAMOS LA VARIABLE contenido CON LA MISMA LONGITUD DEL CONTENIDO
    byte[] contenido = new byte[longitud];

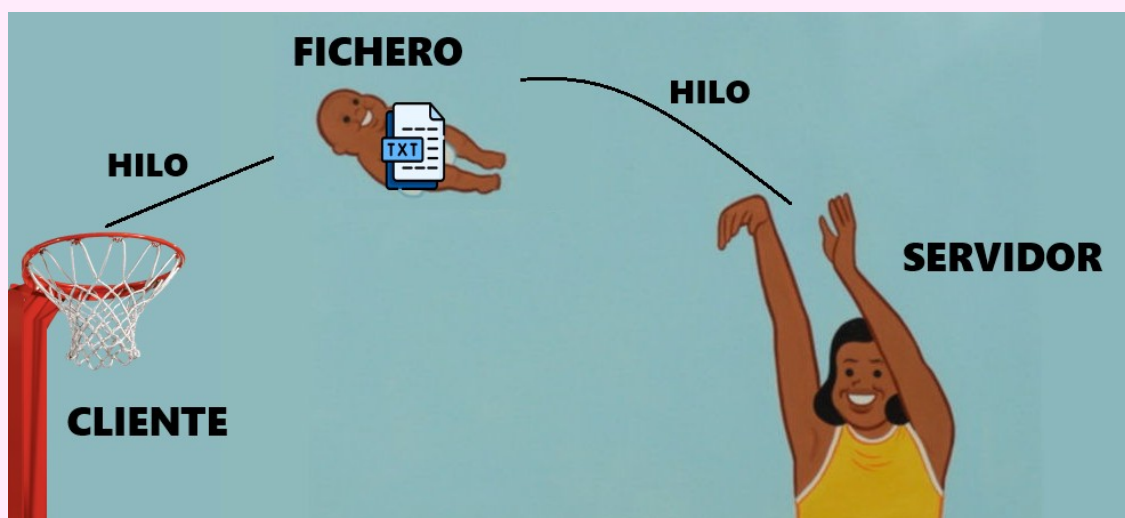
    //RECORREMOS LA VARIABLE contenido ASIGNANDOLE A CADA BYTE EL VALOR LEIDO DE inputStream
    for (int i = 0; i < longitud; i++) {
        contenido[i] = inputStream.readByte();
    }

    //LE MOSTRAMOS AL CLIENTE EL CONTENIDO DEL ARCHIVO
    System.out.println(new String(contenido));

    //SI LA RUTA DEL ARCHIVO NO ES CORRECTA MOSTRAMOS "El archivo no existe" AL CLIENTE
} else {
    System.out.println("El archivo no existe.");
}

//CERRAMOS LA CONEXIÓN
socket.close();

//SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
} catch (IOException ex) {
    Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
}
```



★ Pruebas

Para realizar estas pruebas he creado 4 ficheros .txt llamados xd.txt y los he ubicado en distintas rutas, cada fichero tiene su número, su ruta y una frase ilustre. He hecho todas las solicitudes iniciando una sola vez el servidor.

Servidor

```
Projects x Output x
Actividad3.2 (run) #2 x Actividad3.2 (run) x

run:
El servidor ha iniciado correctamente.
El cliente se ha conectado correctamente
Cliente desconectado
El cliente se ha conectado correctamente
Cliente desconectado
El cliente se ha conectado correctamente
Cliente desconectado
El cliente se ha conectado correctamente
Cliente desconectado
El cliente se ha conectado correctamente
Cliente desconectado
El cliente se ha conectado correctamente
Cliente desconectado
```

```
Projects x Output x
Actividad3.2 (run) x Actividad3.2 (run) #2 x

run:
Introduce la ruta del archivo:
C:\Users\yanet\Desktop\xd.txt

FICHERO DE PRUEBA 1
RUTA: C:\Users\yanet\Desktop\xd.txt

Frase ilustre: "Ponte cachas, mata fachas"

BUILD SUCCESSFUL (total time: 3 seconds)
```

```
Projects x Output x
Actividad3.2 (run) x Actividad3.2 (run) #2 x

run:
Introduce la ruta del archivo:
C:\Users\yanet\Downloads\xd.txt

FICHERO DE PRUEBA 2
RUTA: C:\Users\yanet\Downloads\xd.txt

Frase ilustre: "El mejor facha es el facha muerto"

BUILD SUCCESSFUL (total time: 2 seconds)
```

```
Projects x Output - Actividad3.2 (run) #2 x
Actividad3.2 (run) #2 x

run:
Introduce la ruta del archivo:
C:\PRUEBAS AED\xd.txt

FICHERO DE PRUEBA 3
RUTA: C:\PRUEBAS AED\xd.txt

Frase ilustre: "Machete al machito"

BUILD SUCCESSFUL (total time: 5 minutes 25 seconds)
```

```
Projects x Output - Actividad3.2 (run) #2 x
Actividad3.2 (run) #2 x

run:
Introduce la ruta del archivo:
C:\ACTIVIDADES\Nueva carpeta\xd.txt

FICHERO DE PRUEBA 4
RUTA: C:\ACTIVIDADES\Nueva carpeta\xd.txt

Frase ilustre: "En primaria chivatos,
ahora policias"

BUILD SUCCESSFUL (total time: 2 seconds)
```

Por último lo probamos con una ruta falsa.

```
Projects x Output x
Actividad3.2 (run) #2 x Actividad3.2 (run) x

run:
Introduce la ruta del archivo:
C:\Users\yanet\Desktop\2° DAM\xd.txt
El archivo no existe.

BUILD SUCCESSFUL (total time: 1 minute 28 seconds)
```


★ Conclusiones.

👉 En la **actividad 3.1**, mi conclusión es que la comunicación cliente/servidor es fundamental en el desarrollo de aplicaciones y permite intercambiar información de manera eficiente. En el caso de la aplicación descrita, el cliente y el servidor se comunican a través del puerto 2000. El servidor genera un número secreto aleatorio entre 0 y 100, y el cliente solicita al usuario un número para adivinarlo. El servidor verifica si el número enviado por el cliente es menor, mayor o igual al número secreto, y le indica al cliente qué hacer a continuación.

👉 En la **actividad 3.2**, mi conclusión es que la programación cliente/servidor es una forma eficiente de comunicación entre dos dispositivos. En este ejercicio, pude aprender cómo establecer una conexión entre un cliente y un servidor utilizando un puerto específico. También pude aprender cómo solicitar y recibir un fichero del servidor en el cliente, así como cómo manejar diferentes respuestas del servidor, ya sea enviando el fichero o un mensaje de error.

★ Recursos.

Recursos necesarios para realizar la Tarea:

- ♥ IDE NetBeans.
- ♥ Contenidos de la unidad.
- ♥ Ejemplos expuestos en el contenido de la unidad.

★ Bibliografía.



<https://www.biblia.es/biblia-online.php>



<https://www.churchofjesuschrist.org/study/scriptures?lang=spa>



<https://pastoralsj.org/biblia>



<https://www.biblija.net/biblija.cgi?l=es>



<https://www.bibliatodo.com/la-biblia>