

Tarea PSP04



Indice

• Portada	pg. 1
• Indice	pg. 2
• Introducción	pg. 3
• Actividad 1	pg. 4 - 5
◦ Manual de usuario	pg. 4
◦ Pruebas	pg. 5
• Actividad 2	pg. 6 - 7
◦ Manual de usuario	pg. 6
◦ Pruebas	pg. 7
• Actividad 3	pg. 8 - 14
◦ Manual de usuario	pg. 8 - 13
▪ Diagrama de estados	pg. 8
◦ Pruebas	pg. 14
• Conclusiones	pg. 15
• Recursos	pg. 15
• Bibliografía	pg. 15



Introducción

La tarea 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.

Actividad 4.2.

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

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.



🎀 Actividad 4.1.

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

No he tenido que cambiar nada ya que mi servidor puede trabajar de forma concurrente con varios clientes. El servidor utiliza un bucle while que siempre está a la espera de nuevas conexiones entrantes. Cuando se acepta una nueva conexión, se crea un nuevo objeto de la clase Hilo y se inicializa pasándole el socket de la conexión como parámetro. A continuación, se ejecuta el objeto Hilo mediante el método start().

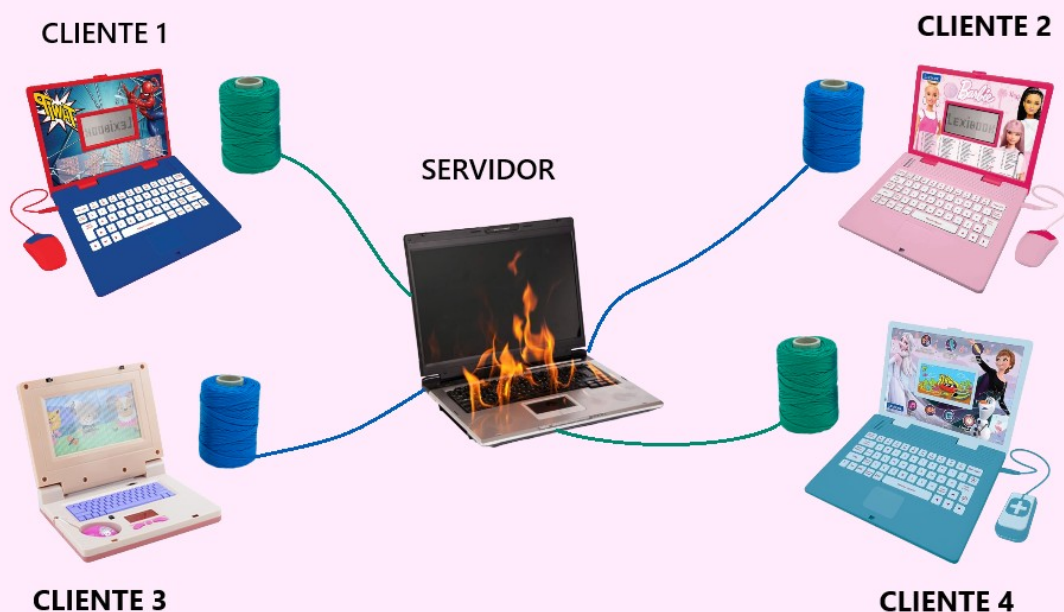
La clase Hilo es una clase personalizada que extiende la clase Thread y se utiliza para manejar las operaciones del cliente individual. Cada vez que se establece una nueva conexión, se crea un nuevo objeto Hilo para manejarla de forma independiente.

En resumen, mi servidor permite trabajar de forma concurrente con varios clientes mediante la creación de hilos que manejan cada conexión entrante de forma independiente, permitiendo así la atención simultánea de múltiples clientes.

```
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);
            }
        }
    }
}
```





Pruebas

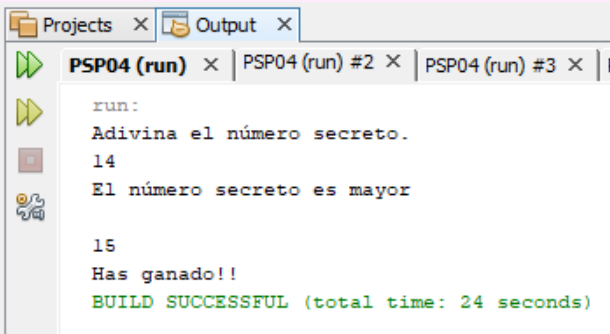
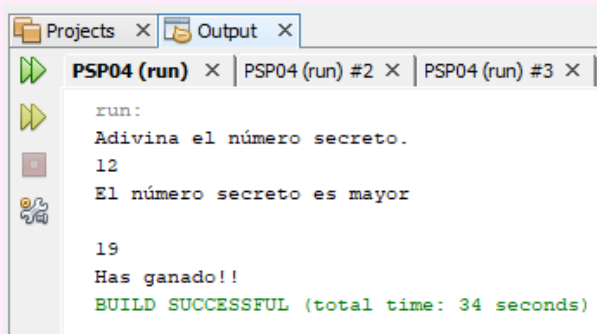
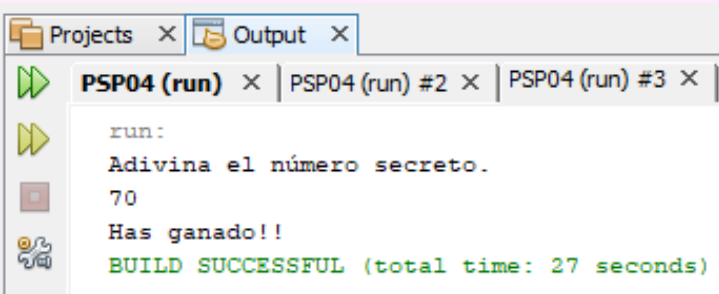
Servidor:

```
Projects x Output x
PSP04 (run) x | PSP04 (run) #2 x | PSP04 (run) #3 x | P

run:
El servidor ha iniciado correctamente.
El cliente se ha conectado correctamente
Numero secreto: 15
El cliente se ha conectado correctamente
Numero secreto: 70
El cliente se ha conectado correctamente
Numero secreto: 19
Número recibido: 14
Número recibido: 15
Cliente desconectado
Número recibido: 70
Cliente desconectado
Número recibido: 12
Número recibido: 19
Cliente desconectado
```



Clientes:

CLIENTE 1	CLIENTE 2
 <p>The screenshot shows the IDE's 'Output' window for Client 1. The window title is 'Projects x Output x'. Below the title bar, there are three tabs: 'PSP04 (run)', 'PSP04 (run) #2', and 'PSP04 (run) #3'. The 'PSP04 (run)' tab is active. The output text is as follows: run: Adivina el número secreto. 14 El número secreto es mayor 15 Has ganado!! BUILD SUCCESSFUL (total time: 24 seconds)</p>	 <p>The screenshot shows the IDE's 'Output' window for Client 2. The window title is 'Projects x Output x'. Below the title bar, there are three tabs: 'PSP04 (run)', 'PSP04 (run) #2', and 'PSP04 (run) #3'. The 'PSP04 (run)' tab is active. The output text is as follows: run: Adivina el número secreto. 12 El número secreto es mayor 19 Has ganado!! BUILD SUCCESSFUL (total time: 34 seconds)</p>
CLIENTE 3	
 <p>The screenshot shows the IDE's 'Output' window for Client 3. The window title is 'Projects x Output x'. Below the title bar, there are three tabs: 'PSP04 (run)', 'PSP04 (run) #2', and 'PSP04 (run) #3'. The 'PSP04 (run)' tab is active. The output text is as follows: run: Adivina el número secreto. 70 Has ganado!! BUILD SUCCESSFUL (total time: 27 seconds)</p>	

🎀 Actividad 4.2.

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

Tampoco he tenido que cambiar nada ya que este servidor también puede trabajar de forma concurrente con varios clientes. El servidor utiliza un bucle while que siempre está a la espera de nuevas conexiones entrantes. Cuando se acepta una nueva conexión, se crea un nuevo objeto de la clase Hilo y se inicializa pasándole el socket de la conexión como parámetro. A continuación, se ejecuta el objeto Hilo mediante el método start().

La clase Hilo es una clase personalizada que extiende la clase Thread y se utiliza para manejar las operaciones del cliente individual. Cada vez que se establece una nueva conexión, se crea un nuevo objeto Hilo para manejarla de forma independiente.

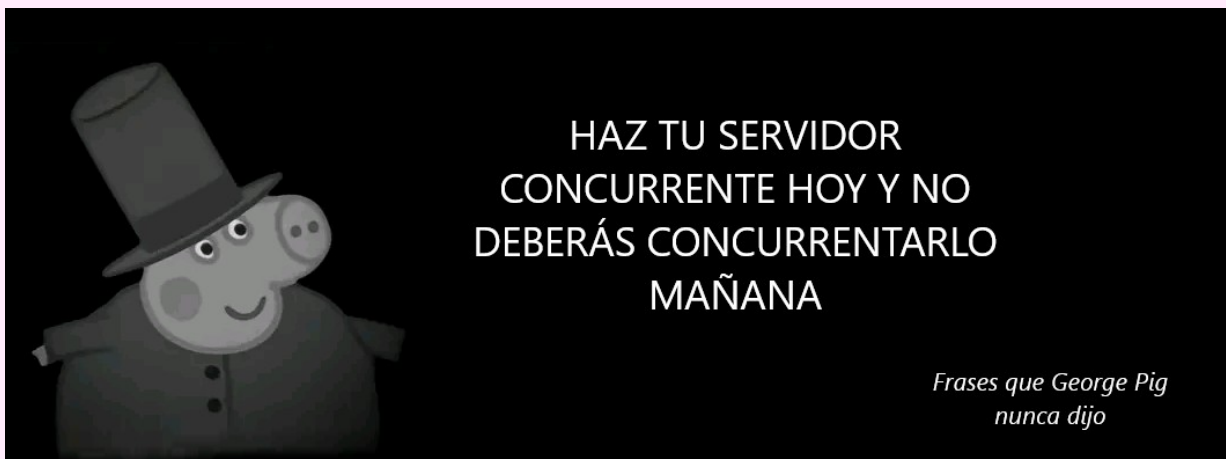
En resumen, este servidor también permite trabajar de forma concurrente con varios clientes mediante la creación de hilos que manejan cada conexión entrante de forma independiente, permitiendo así la atención simultánea de múltiples clientes.

```
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 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);
        }
    }
}
```





Pruebas

Servidor:

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

Clientes:

CLIENTE 1	CLIENTE 2
<pre>run: Introduce la ruta del archivo: C:\Users\yanet\Desktop\xd.txt FICHERO DE PRUEBA 1 RUTA: C:\Users\yanet\Desktop\xd.txt El comunismo no priva a nadie del poder de apropiarse productos sociales; lo único que no admite es el poder de usurpar por medio de esta apropiación el trabajo ajeno. BUILD SUCCESSFUL (total time: 17 seconds)</pre>	<pre>run: Introduce la ruta del archivo: C:\Users\yanet\Desktop\DAM\xd.txt FICHERO DE PRUEBA 2 RUTA: C:\Users\yanet\Desktop\DAM\xd.txt ¿Hubo nunca, en cualquier época, en cualquier país, un solo ejemplo de clase privilegiada y dominante que hiciera concesiones de forma libre y espontáneamente sin ser obligada a ella por la fuerza o por el miedo? BUILD SUCCESSFUL (total time: 22 seconds)</pre>
CLIENTE 3	
<pre>run: Introduce la ruta del archivo: C:\RutaInexistente El archivo no existe. BUILD SUCCESSFUL (total time: 39 seconds)</pre>	

🎀 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.

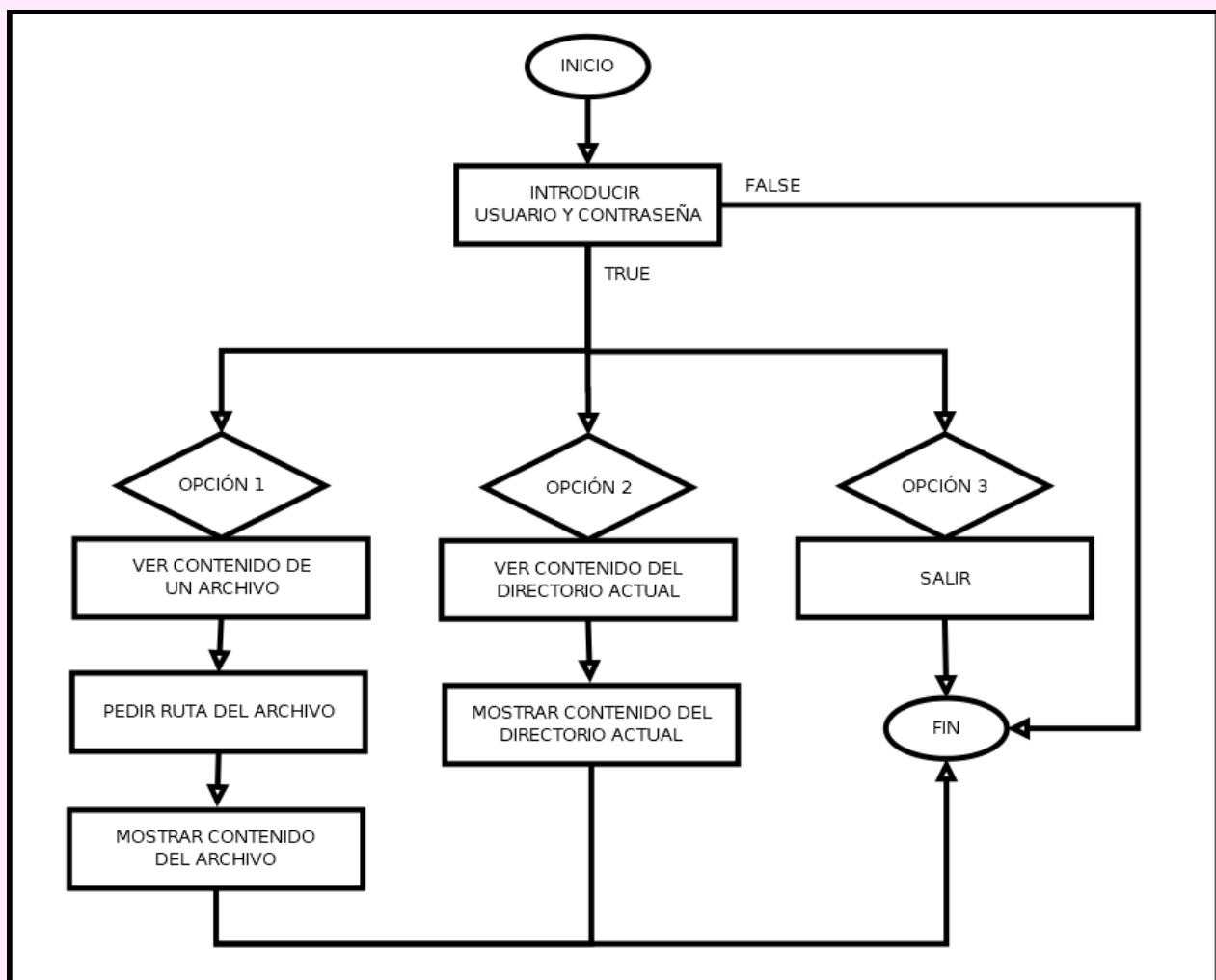
DIAGRAMA DE ESTADOS

Este diagrama de estado muestra el funcionamiento del servidor, primero se inicia, pide el usuario y la contraseña, si no son correctos finaliza el programa, si son correctos da tres opciones.

OPCIÓN 1 VER CONTENIDO DE UN ARCHIVO: El servidor pide la ruta del archivo, muestra el contenido de este archivo y finaliza el programa.

OPCIÓN 2 VER CONTENIDO DEL DIRECTORIO ACTUAL: El servidor muestra el contenido del directorio actual y finaliza el programa.

OPCIÓN 3 SALIR: El servidor finaliza el programa.



En esta actividad explicare solamente el nuevo código añadido, por motivos evidentes xd. Lo primero que haré será modificar la clase Cliente, y teniendo en cuenta los cambios realizados modificaré el hilo.

Lo primero que haremos en la clase cliente será añadir las siguientes lineas, estas piden al usuario el usuario y contraseña y los escribe en el flujo de salida.

```
//GUARDAMOS EN "mensaje" EL STRING QUE SOLICITA EL USUARIO
String mensaje = inputStream.readUTF().trim();
//MOSTRAMOS "mensaje" POR PANTALLA
System.out.println(mensaje);
//GUARDAMOS EN UN STRING EL USUARIO
String usuario = scn.next();
//ESCRIBIMOS EL USUARIO EN EL FLUJO DE SALIDA
ouputStream.writeUTF(usuario);

//GUARDAMOS EN "mensaje" EL STRING QUE SOLICITA LA CONTRASEÑA
mensaje = inputStream.readUTF().trim();
//MOSTRAMOS "mensaje" POR PANTALLA
System.out.println(mensaje);
//GUARDAMOS EN UN STRING LA CONTRASEÑA
String contrasenia = scn.next();
//ESCRIBIMOS LA CONTRASEÑAEN EL FLUJO DE SALIDA
ouputStream.writeUTF(contrasenia);
```

Ahora creamos un booleano que leerá del flujo de entrada si el cliente se ha logueado con éxito o no, y si es así, creamos un booleano para finalizar el programa y un bucle while que se ejecutara mientras “salir” sea false, en él leemos del flujo de entrada las opciones y las mostramos por pantalla, guardamos en “opcion” la opción seleccionada y la escribimos en el flujo de salida.

En resumen, se crea un booleano que leerá del flujo de entrada si el cliente se ha logueado con éxito.

```
//CREAMOS UN BOOLEANO QUE NOS INDICA SI EL CLIENTE SE HA LOGUEADO
//(SI SON CORRECTOS EL USUARIO Y CONTRASEÑA)
boolean exito = inputStream.readBoolean();

//SI SON CORRECTOS EL USUARIO Y CONTRASEÑA
if(exito){
    //CREAMOS UN BOOLEANO PARA FINALIZAR EL PROGRAMA
    boolean salir = false;

    //MIENTRAS "salir" SEA FALSO
    while (!salir) {
        //LEEMOS EL MENSAJE QUE CONTIENDRÁ LAS OPCIONES
        mensaje = inputStream.readUTF().trim();
        //LAS MOSTRAMOS POR PANTALLA
        System.out.println(mensaje);
        //GUARDAMOS EN opcion LA OPCION SELECCIONADA
        String opcion = scn.next();
        //ESCRIBIMOS EN EL FLUJO DE SALIDA LA OPCION
        ouputStream.writeUTF(opcion);
    }
}
```



Opción 1 (VER CONTENIDO DE UN ARCHIVO): Primero pedimos la ruta y la escribimos en el flujo de salida, creamos un “if” que comprobará si la ruta existe o no. Si la ruta existe leemos la longitud del contenido del archivo, creamos la variable “contenido” con la misma longitud del contenido, recorremos la variable “contenido” asignándole a cada byte el valor leído y le mostramos al cliente el contenido del archivo. Si no existe mostramos "El archivo no existe" al cliente. Por último pasamos “salir” a true finalizando el programa.

```
if(opcion.equals("1")){
    //LE PEDIMOS AL CLIENTE LA RUTA DEL ARCHIVO EJ: C:\Users\yanet\Desktop\xd.txt
    System.out.println("Introduce la ruta del archivo: ");
    String ruta = scn.next();

    //ESCRIBIMOS LA RUTA EN EL FLUJO DE SALIDA
    ouputStream.writeUTF(ruta);

    //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.");}
    //FINALIZAMOS EL PROGRAMA
    salir = true;
}
```

Opción 2 (VER CONTENIDO DEL DIRECTORIO ACTUAL): Leemos del flujo de entrada el contenido.length y lo almacenamos en “numContenido”, hacemos un bucle for que se ejecutará “numContenido” numero de veces, cada vez que se ejecute almacenará en “nombre” el nombre de un fichero o carpeta y lo mostrará por pantalla, por último pasamos “salir” a true finalizando el programa.

```
}else if(opcion.equals("2")){
    //LEEMOS EN EL FLUJO DE ENTRADA EL NÚMERO DE FICHEROS Y CARPETAS QUE HAY
    int numContenido = inputStream.readInt();
    //CREAMOS UN BUCLE FOR QUE EJECUTARÁ "numContenido" VECES
    for (int i = 0; i < numContenido; i++) {
        //LEE DEL FLUJO DE ENTRADA EL NOMBRE DE UN FICHERO O CALPETA Y LO ALMACENA EN "nombre"
        String nombre = inputStream.readUTF().trim();
        //MUESTRA EL NOMBRE POR PANTALLA
        System.out.println(nombre);
    }
    //FINALIZAMOS EL PROGRAMA
    salir = true;
}
```

Opción 3 (SALIR): Si se inserta algo que no sea 1 o 2 pasamos “salir” a true finalizando el programa. Si “exito” es false (el cliente no se ha logueado con éxito) se muestra “USUARIO O CONTRASEÑA INCORRECTOS” y finaliza el programa.

```
    }else{
        //FINALIZAMOS EL PROGRAMA
        salir = true;
    }
}
}else{
    //COMUNICAMOS AL CLIEITE QUE LOS DATOS DE LOGUI SON INCORRECTOS Y FINALIZAMOS EL PROGRAMA
    System.out.println("USUARIO O CONTRASEÑA INCORRECTOS");
}
```

Teniendo en cuenta la clase “Cliente” modificaremos la clase “Hilo”, lo primero que haremos será escribir en el flujo de salida el mensaje para solicitar el usuario, leemos en el flujo de entrada el usuario y lo almacenamos en "usuario", escribimos en el flujo de salida el mensaje para solicitar la contraseña y leemos en el flujo de entrada la contraseña y la almacenamos en "contrasenia".

```
//ESCRIBIMOS EN EL FLUJO DE SALIDA EL MENSAJE PARA SOLICITAR EL USUARIO
ouputStream.writeUTF("USUARIO:");
//LEEMOS EN EL FLUJO DE ENTRADA EL USUARIO Y LO ALMACENAMOS EN "usuario"
String usuario = inputStream.readUTF().trim();

//ESCRIBIMOS EN EL FLUJO DE SALIDA EL MENSAJE PARA SOLICITAR LA CONTRASEÑA
ouputStream.writeUTF("CONTRASEÑA:");
//LEEMOS EN EL FLUJO DE ENTRADA LA CONTRASEÑA Y LA ALMACENAMOS EN "contrasenia"
String contrasenia = inputStream.readUTF().trim();
```

Si el usuario es "javier" y la contraseña es "secreta", escribimos en el flujo de salida que el cliente se ha logueado con exito, escribimos en el flujo de salida el mensaje con las opciones, leemos en el flujo de entrada la opción seleccionada, la almacenamos en "opcion" y creamos un booleano para finalizar el programa.

```
//SI EL USUARIO ES "javier" Y LA CONTRASEÑA ES "secreta"
if(usuario.equals("javier") && contrasenia.equals("secreta")){
    //ESCRIBIMOS EN EL FLUJO DE SALIDA QUE EL CLIENTE SE HA LOGUEADO CON EXITO
    ouputStream.writeBoolean(true);
    //ESCRIBIMOS EN EL FLUJO DE SALIDA EL MENSAJE CON LAS OPCIONES
    ouputStream.writeUTF("\n1 VER CONTENIDO DE UN ARCHIVO"
        + "\n2 VER CONTENIDO DEL DIRECTORIO ACTUAL"
        + "\n3 Salir");
    //LEEMOS EN EL FLUJO DE ENTRADA LA OPCIÓN SELECCIONADA Y LA ALMACENAMOS EN "opcion"
    String opcion = inputStream.readUTF().trim();
    //CREAMOS UN BOOLEANO PARA FINALIZAR EL PROGRAMA
    boolean salir = false;
```

Si el usuario no es "javier" o la contraseña no es "secreta" escribimos en el flujo de salida que el cliente no se ha logueado con éxito.

```
}else{
    //ESCRIBIMOS EN EL FLUJO DE SALIDA QUE EL CLIENTE NO SE HA LOGUEADO CON EXITO
    ouputStream.writeBoolean(false);
}
```

Ahora creamos un bucle while el cual se ejecutará mientras salir sea falso.

Opción 1 (VER CONTENIDO DE UN ARCHIVO): Leemos del flujo de entrada la ruta, la almacenaremos en el String “ruta” y crearemos el objeto File usando esa ruta.

Creamos un “if” para comprobar si la ruta existe, si la ruta no existe lo escribimos en el flujo de salida y finalizamos el programa.

Si la ruta 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 y le añadimos la línea leída a “contenido” Junto con el carácter 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” y escribimos en el flujo de salida el valor de cada uno. Por último cerramos la conexión y lo comunicamos en el servidor.

```
while(!salir){  
    if(opcion.equals("1")){  
        //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);  
  
        //SI LA RUTA EXISTE EJECUTAMOS EL CÓDIGO  
        if(file.exists()){  
            //ESCRIBIMOS EN EL FLUJO DE SALIDA QUE LA RUTA SI EXISTE  
            outputStream.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  
            outputStream.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  
                outputStream.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{  
            outputStream.writeBoolean(false);  
        }  
        //FINALIZAMOS EL PROGRAMA  
        salir = true;  
    }  
}
```

Opción 2 (VER CONTENIDO DEL DIRECTORIO ACTUAL): creamos un objeto File que representa la ruta actual, guardamos en "contenido" los archivos y carpetas, escribimos en el flujo de salida el número de archivos y carpetas que hay, escribimos en el flujo de salida uno a uno el nombre de cada archivo y carpeta y finalizamos el programa.

```
}else if(opcion.equals("2")){  
    //CREAMOS UN OBJETO File QUE REPRESENTA LA RUTA ACTUAL  
    File directorioActual = new File("./");  
    //GUARDAMOS EN "contenido" LOS ARCHIVOS Y CARPETAS  
    File[] contenido = directorioActual.listFiles();  
  
    //ESCRIBIMOS EN EL FLUJO DE SALIDA EL NÚMERO DE ARCHIVOS Y CARPETAS QUE HAY  
    outputStream.writeInt(contenido.length);  
  
    //ESCRIBIMOS EN EL FLUJO DE SALIDA UNO A UNO EL NOMBRE DE CADA ARCHIVO Y CARPETA  
    for (int i = 0; i < contenido.length; i++) {  
        outputStream.writeUTF(contenido[i].getName());  
    }  
    //FINALIZAMOS EL PROGRAMA  
    salir = true;  
}
```

Opción 3 (SALIR): Si el cliente escoge la opción 3 o escribe cualquier cosa que no sea "1" o "2" el programa finalizará.

```
//SI EL CLIENTE ESCOJE LA OPCIÓN 3 O ESCRIBE CUALQUIER COSA QUE NO SEA  
//"1" o "2" EL PROGRAMA FINALIZARÁ  
}else{  
    //FINALIZAMOS EL PROGRAMA  
    salir = true;  
}
```





Pruebas

1. Prueba usando un usuario o contraseña **incorrectos**:

```
run:
USUARIO:
Leñador
CONTRASEÑA:
69
USUARIO O CONTRASEÑA INCORRECTOS
BUILD SUCCESSFUL (total time: 16 seconds)
```

```
run:
USUARIO:
javier
CONTRASEÑA:
secreto
USUARIO O CONTRASEÑA INCORRECTOS
BUILD SUCCESSFUL (total time: 16 seconds)
```

2. Prueba usando un usuario o contraseña **correctos** + **opción salir**:

```
run:
USUARIO:
javier
CONTRASEÑA:
secreta
1 VER CONTENIDO DE UN ARCHIVO
2 VER CONTENIDO DEL DIRECTORIO ACTUAL
3 Salir
3
BUILD SUCCESSFUL (total time: 10 seconds)
```

```
run:
USUARIO:
javier
CONTRASEÑA:
secreta
1 VER CONTENIDO DE UN ARCHIVO
2 VER CONTENIDO DEL DIRECTORIO ACTUAL
3 Salir
69
BUILD SUCCESSFUL (total time: 8 seconds)
```

3. Prueba usando un usuario o contraseña **correctos** + :

Opción ver contenido directorio actual

```
run:
USUARIO:
javier
CONTRASEÑA:
secreta
1 VER CONTENIDO DE UN ARCHIVO
2 VER CONTENIDO DEL DIRECTORIO ACTUAL
3 Salir
2
build
build.xml
CarpetaEjemplo
ejemplol1.txt
manifest.mf
nbproject
src
test
BUILD SUCCESSFUL (total time: 13 seconds)
```

Opción ver contenido de un archivo

```
run:
USUARIO:
javier
CONTRASEÑA:
secreta
1 VER CONTENIDO DE UN ARCHIVO
2 VER CONTENIDO DEL DIRECTORIO ACTUAL
3 Salir
1
Introduce la ruta del archivo:
C:\Users\yanet\Desktop\xd.txt

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

El opresor no seria tan fuerte si no tuviese
cómplices entre los propios oprimidos.

BUILD SUCCESSFUL (total time: 10 seconds)
```




Conclusiones.

👉 En el **ejercicio 1**, mi conclusión ha sido soy una crack por haber hecho el servidor multihilo desde el principio.

👉 En el **ejercicio 2**, mi conclusión ha sido la misma que en el ejercicio 1.

👉 En el **ejercicio 3**, mi conclusión ha sido que se puede crear un servidor que funcione como un sistema de autenticación básico utilizando un nombre de usuario y contraseña específicos. Una vez que se inicie sesión correctamente, el sistema permitirá al usuario ver el contenido del directorio actual, mostrar el contenido de archivos específicos y salir del servidor. Esto proporciona una forma básica de gestión de archivos y acceso controlado a través de un servidor.



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>