ACTIVIDAD 4.1

Modificamos la actividad 1 del tema 3 para hacer que el servidor trabaje con varias conexiones de clientes de manera concurrente. Para ello introduciré 2 modificaciones en el proyecto:

1. Dentro de la clase Servidor extendemos la clase Thread y sobrescribimos el método Run que implementa:

```
@Override
public void run() {
    //Declaración de variables
    int numeroAleatorio, numUsuario;
    short valor = 3;
    //Genera número aleatorio del 0 al 100 y lo muestra por pantalla
    numeroAleatorio = (int) (Math.random() * 100);
    System.out.println("Numero aleatorio: " + numeroAleatorio);
    do {
        try {
            //Creamos un flujo de entrada del socket y leemos el entero de entrada
            DataInputStream flujoEntrada = new DataInputStream(sCliente.getInputStream());
            numUsuario = flujoEntrada.readInt();

            //Comparamos números para saber si es mayor, menor o igual
            if (numUsuario < numeroAleatorio) {
                valor = 2;
            } else if (numUsuario > numeroAleatorio) {
                valor = 1;
            } else {
                valor = 0;
/                 flujoEntrada.close();
            }
            DataOutputStream flujoSalida = new DataOutputStream(sCliente.getOutputStream());
            flujoSalida.writeShort(valor);

        } catch (IOException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }

    } while (valor > 0);//Repetimos bucle hasta que el valor sea 0 (correcto)
    try {
        sCliente.close();
    } catch (IOException ex) {
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Dentro del método vamos meter el código que realizará las operaciones de la clase

En el método main() ya podemos crear un hilo de esta clase mediante el método star(). El propio hilo se encargará ya de gestionar las conexiones de entrada de los clientes:

```
public static void main(String[] args) {
    int numCliente = 0;
    try {
        //Crea el servicio de socket servidor con los datos
        ServerSocket socketServidor = new ServerSocket(Puerto);
        System.out.println("Esperando peticion de cliente...");
        //En espera de petición de cliente
        while (true) {
            numCliente++;
            Socket socketCliente = socketServidor.accept();
            System.out.println("\nConectado cliente " + numCliente);
            new Servidor(socketCliente).start();
        }
    } catch (IOException ex) {
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Creamos hilos de manera infinita para que acepte un número ilimitado de conexiones de clientes

2. Dentro de la clase Cliente también vamos a modificar el método main() para hacer que se ejecute 3 veces la clase mediante un bucle, para probar la concurrencia de conexiones entre Cliente-Servidor:

```
public static void main(String[] args) {
    int numCliente = 0;
    do {
        numCliente++;
        new Cliente(numCliente);
    } while (numCliente < 3);
}
```

He decidido a modo de prueba que se genere 3 veces la clase pero podríamos poner el número que queramos

PRUEBAS DE APLICACIÓN:

Abrimos la terminal de comandos, vamos donde tenemos el proyecto, compilamos las 2 clases y ejecutamos la clase Servidor que se quedará en espera de conexión de cliente. Una vez recibe las conexiones muestra por pantalla que cliente se ha conectado y que número aleatorio ha generado. Así podemos ver que se han conectado 3 clientes y se queda esperando más conexiones si las hubiera:

```
C:\Users\DANILOR\OneDrive - Educacyl\Asignaturas 2ºDAM\PSP\Tareas\Tarea 4\Tarea_4_PSP\src\actividad_4_1>javac Servidor.java

C:\Users\DANILOR\OneDrive - Educacyl\Asignaturas 2ºDAM\PSP\Tareas\Tarea 4\Tarea_4_PSP\src\actividad_4_1>javac Cliente.java

C:\Users\DANILOR\OneDrive - Educacyl\Asignaturas 2ºDAM\PSP\Tareas\Tarea 4\Tarea_4_PSP\src\actividad_4_1>cd..

C:\Users\DANILOR\OneDrive - Educacyl\Asignaturas 2ºDAM\PSP\Tareas\Tarea 4\Tarea_4_PSP\src>java actividad_4_1.Servidor
Esperando peticion de cliente...

Conectado cliente 1
Numero aleatorio: 54

Conectado cliente 2
Numero aleatorio: 77

Conectado cliente 3
Numero aleatorio: 99
```

Por otro lado hemos abierto otra terminal, vamos donde tenemos el proyecto y ejecutamos la clase Cliente. Nada más arrancar se conectará al servidor y nos va a solicitar números hasta que acertemos cual es el que ha generado el Servidor. Esto lo hará sucesivamente durante 3 veces, actuando como 3 clientes distintos:

```
C:\Users\DANILOR\OneDrive - Educacyl\Asignaturas 2ºDAM\PSP\Tareas\Tarea 4\Tarea_4_PSP\src>java actividad_4_1.Cliente
Cliente 1

ADIVINE EL NUMERO DEL 0 AL 100 GENERADO POR EL SERVIDOR
-------------------------------------------------------

Intento 1:
50
-- ES MAYOR --

Intento 2:
60
-- ES MENOR --

Intento 3:
54
ENHORABUENA, HAS ACERTADO EL NUMERO SECRETO !!!!!

Cliente 2

ADIVINE EL NUMERO DEL 0 AL 100 GENERADO POR EL SERVIDOR
-------------------------------------------------------

Intento 1:
70
-- ES MAYOR --

Intento 2:
80
-- ES MENOR --

Intento 3:
77
ENHORABUENA, HAS ACERTADO EL NUMERO SECRETO !!!!!

Cliente 3

ADIVINE EL NUMERO DEL 0 AL 100 GENERADO POR EL SERVIDOR
-------------------------------------------------------

Intento 1:
0
-- ES MAYOR --

Intento 2:
100
-- ES MENOR --

Intento 3:
99
```

Observamos que se conectan los 3 clientes de manera recurrente ejecutándose la aplicación completa para cada uno

ACTIVIDAD 4.2

De la misma manera que en el ejercicio anterior, he modificado varias partes del proyecto para hacer que el Servidor acepte conexiones del cliente de manera recurrente:

1. A la clase servidor le extendemos la clase Thread e implementamos y sobrescribimos el método run() de la interfaz, para posteriormente en poder ejecutar como un hilo dentro del método main():

```
public class Servidor_2 extends Thread {
```

```
public static void main(String[] args) {
    int numCliente=0;
    try {
        //Creamos socket del servidor con el puerto 1500
        ServerSocket socketServidor = new ServerSocket(Puerto);
        System.out.println("Esperando peticion de cliente...");

        while (true) {
            //Espera la peticion de cliente y cuando entra la acepta
            Socket socketCliente = socketServidor.accept();
            numCliente++;
            System.out.println("Cliente "+numCliente+" conectado");
            new Servidor_2(socketCliente).start();
```

```
@Override
public void run() {
```

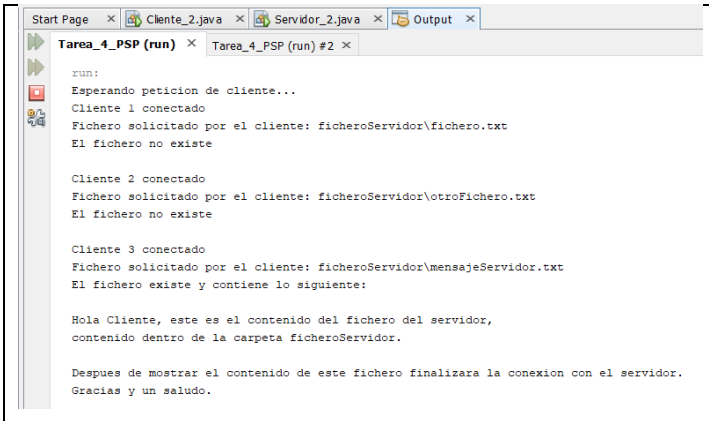
Este último método va a contener el código de la lógica principal de la aplicación servidor

2. En la clase Cliente dentro del método main() vamos a añadir un bucle de 3 repeticiones para que en cada una se haga una instancia de la propia clase y así poder probar que funciona la concurrencia de conexiones con el servidor de manera correcta:

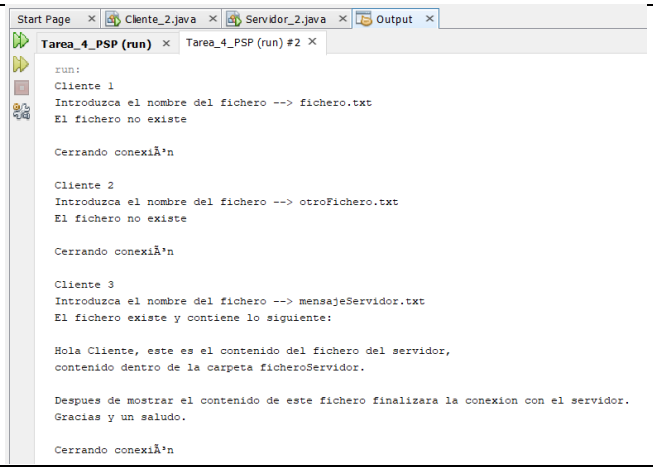
```
public static void main(String[] args) {
    int numCliente=0;

    do {
        numCliente++;
        new Cliente_2(numCliente);
    } while (numCliente<3);
```

PRUEBAS DE APLICACIÓN:

Para realizar las pruebas de esta aplicación voy a utilizar el propio IDE NetBeans en vez de la terminal de comandos, para comprobar que tenemos distintas maneras de hacerlo y todas son igual de válidas. Lo primero que haremos será ejecutar la clase Servidor para a continuación ejecutar la clase Cliente que se nos abrirá en una ventana distinta y trabajarán de manera simultánea. He realizado esta simulación en la parte del Servidor se van generando los mensajes que nos dan información de los clientes que se conectan y cuando se desconectan; a su vez en la parte del cliente vamos a conectarnos al servidor y va a solicitar el nombre de un fichero:



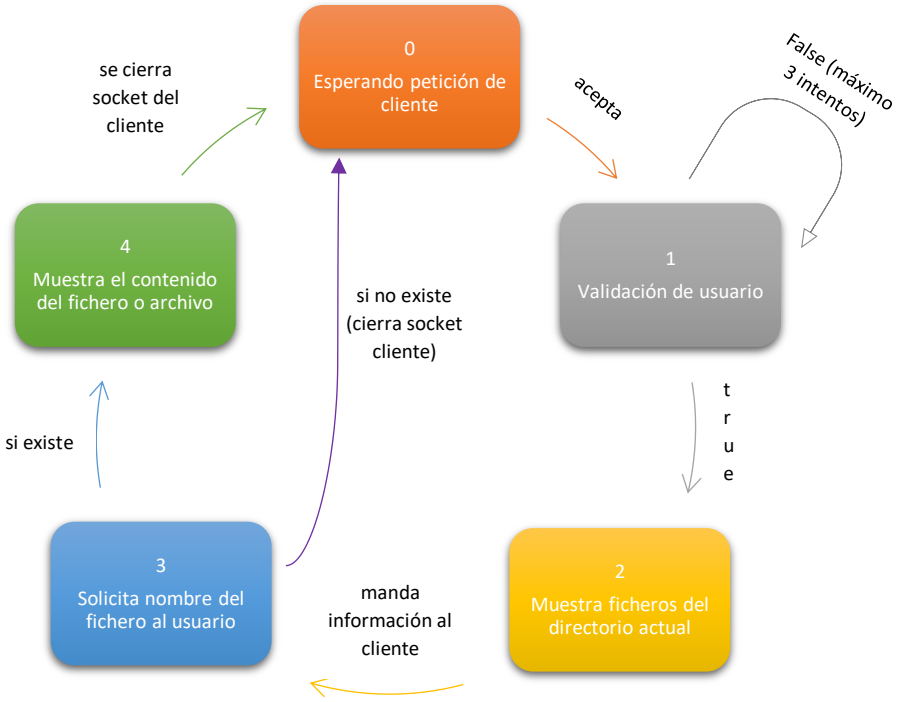
*En la parte de **Servidor** he creado dentro de la carpeta ficheroServidor un txt llamado mensajeServidor a modo prueba*



*En la parte **Cliente** vamos a solicitar 2 ficheros erróneos y uno que si existe a modo prueba*

ACTIVIDAD 4.3

DIAGRAMA DE FLUJO DEL SERVIDOR:



Nuevamente para realizar este ejercicio necesitamos una clase que haga de servidor y otra clase que haga de cliente:

- *Servidor* extendemos la clase Thread junto con la definición del método run para poder crear hilos de ejecución y así aceptar varias conexiones de cliente de manera concurrente.

```
public class Servidor_3 extends Thread {
```

```
@Override
public void run() {
    int intentos = 0;
    boolean valida = false;

    try {
        //Crea un flujo de entrada del socket para recoger el fichero enviado por el cliente
        DataInputStream flujoEntrada = new DataInputStream(socketCli.getInputStream());
        //Creamos un flujo de salida del socket
        DataOutputStream flujoSalida = new DataOutputStream(socketCli.getOutputStream());
        //Permite 3 intentos erróneos de nombre de usuario y contraseña
        do {
            intentos++;
            valida = validacionUsuCon(valida, flujoEntrada, flujoSalida);
        } while (intentos < 3 && valida == false);

        if (valida) {//Si usuario y contraseña son correctos
            System.out.println("¡¡Bienvenido!!");
            mostrarDirectorioActual(flujoSalida);
            //Recoge el nombre del fichero solicitado por el usuario
            nombreUsuarioFichero = flujoEntrada.readUTF();
            ficheroServidor = new File(nombreUsuarioFichero);

            if (ficheroServidor.exists()) {//Comprueba si el fichero solicitado existe
                if (ficheroServidor.isFile()) {//Comprobamos si es un fichero
                    flujoSalida.writeInt(1);
                    mostrarFichero(flujoSalida);
                } else if (ficheroServidor.isDirectory()) {//Comprobamos si es un directorio
                    flujoSalida.writeInt(2);
                    mostrarArchivo(flujoSalida);
                }
                flujoEntrada.close();

            } else {
                flujoSalida.writeInt(0);
                flujoSalida.writeUTF("No existe un fichero con ese nombre");
            }
```

Contiene la lógica principal de la aplicación junto con las llamadas a otros métodos

Para hacer un código más limpio estructurado, organizado y modular he creado 4 métodos a mayores para cada una de las tareas más importantes:

```
private static boolean validacionUsuCon(boolean valida, DataInputStream flujoEntrada, DataOutputStream flujoOut) throws IOException {
```

```
private void mostrarFichero(DataOutputStream flujoSalida) {
```

```
private void mostrarArchivo(DataOutputStream flujoSalida) {
```

```
private void mostrarDirectorioActual(DataOutputStream flujoSalida) throws IOException {
```

- *Cliente* dentro de esta clase definimos el constructor, el método main y creamos 2 métodos a mayores, como en la clase Servidor para poder organizar mejor el código:

```
public Cliente_3(int numCliente) throws IOException {
    //Declaración de variables y objetos
    Scanner teclado = new Scanner(System.in);
    boolean valida;

    //Creamos la conexión al servidor
    Socket socketCliente = new Socket(Host, Puerto);
    System.out.println("Cliente " + numCliente);

    //Creamos un flujo de salida del Socket
    DataOutputStream flujoSalida = new DataOutputStream(socketCliente.getOutputStream());
    //Creamos un flujo de entrada del socket
    DataInputStream flujoEntrada = new DataInputStream(socketCliente.getInputStream());
    //Compruebe si el usuario y contraseña son válidos
    valida = peticionUsuCon(false, flujoEntrada, flujoSalida);
    if (valida) {
        mostrarFicheroCliente(flujoEntrada, flujoSalida, teclado);
        System.out.println("\nCerrando conexión\n");
        socketCliente.close();
    } else {
        System.out.println("El cliente " + numCliente + " ha excedido el número máximo de intentos\n");
    }
```

Dentro del constructor creamos los flujos de salida/entrada con servidor y de entrada de teclado

```
private static boolean peticionUsuCon(boolean valida, DataInputStream flujoEntrada, DataOutputStream flujoSalida) throws IOException {
```

```
private void mostrarFicheroCliente(DataInputStream flujoEntrada, DataOutputStream flujoSalida, Scanner teclado) {
```

Métodos para solicitar usuario y contraseña al usuario y método para mostrar por pantalla el fichero/archivo

PRUEBAS DE APLICACIÓN:

1. Dentro de nuestro proyecto en NetBeans ejecutamos la clase Servidor y a continuación la clase Cliente, se inicia la aplicación solicitando usuario y contraseña, cada cliente le asigno un número para distinguirlos:

Cliente	Servidor
run: Cliente 1 Nombre de usuario: dani Contraseña de usuario: 1234 Usuario o contraseña no válida Nombre de usuario: javier Contraseña de usuario: secreta Usuario y contraseña correctas El directorio actual contiene los siguientes ficheros y archivos --- build build.xml ficheroServidor manifest.mf nbproject src test --- Introduzca el nombre del fichero que desea ver -->	Start Page × Output × Servidor_3.java × Cliente_3.java × Tarea_4_PSP (run) × Tarea_4_PSP (run) #2 × run: Esperando peticion de cliente... Cliente 1 conectado ¡¡Bienvenido!! Mandando ficheros que contiene el proyecto al cliente
Probamos la validación metiendo otro usuario y contraseña distinto a “javier”, “secreta” para a continuación mostrar contenido del proyecto	Mensaje de bienvenida y de información de tarea del servidor

2. Como ejemplo solicitamos el contenido de la carpeta “src” para que nos lo muestre por pantalla:

```
El directorio actual contiene los siguientes ficheros y archivos
----------------------------------------------------------------------
build
build.xml
ficheroServidor
manifest.mf
nbproject
src
test
----------------------------------------------------------------------
Introduzca el nombre del fichero que desea ver --> src
El archivo existe y contiene:

----------------------------------------------------------------------
actividad_4_1
actividad_4_2
actividad_4_3
----------------------------------------------------------------------

Cerrando conexión

Cliente 2
Nombre de usuario:
```

Después de mostrarnos el contenido de la carpeta se cierra la conexión con el servidor del cliente número 1

3. Ya con el cliente número 2 vamos a solicitar un fichero, he elegido como ejemplo “manifest.mf”:

```
Cliente 2
Nombre de usuario:
javier
Contraseña de usuario:
secreta
Usuario y contraseña correctas

El directorio actual contiene los siguientes ficheros y archivos
----------------------------------------------------------------------
build
build.xml
ficheroServidor
manifest.mf
nbproject
src
test
----------------------------------------------------------------------
Introduzca el nombre del fichero que desea ver --> manifest.mf
El fichero existe y contiene:


----------------------------------------------------------------------
Manifest-Version: 1.0
X-COMMENT: Main-Class will be added automatically by build

----------------------------------------------------------------------

Cerrando conexión
```

Comprobamos que nos muestra por pantalla el contenido del fichero

4. Por último con el cliente 3 introducimos el nombre de un fichero que no existe dentro del proyecto:

```
Cliente 3
Nombre de usuario:
javier
Contraseña de usuario:
secreta
Usuario y contraseña correctas

El directorio actual contiene los siguientes ficheros y archivos
------------------------------------------------------------------------
build
build.xml
ficheroServidor
manifest.mf
nbproject
src
test
------------------------------------------------------------------------
Introduzca el nombre del fichero que desea ver --> nombreFichero
No existe un fichero con ese nombre

Cerrando conexión
```

La aplicación comprueba que el fichero no existe y nos devuelve un mensaje informándonos y cerrando la aplicación

Además de haber comprobado que verifica usuario y contraseña y de que también verifica y muestra los ficheros dentro del contenido, podemos ver que la aplicación ha sido capaz de conectar 3 clientes de manera concurrente al servidor sin errores y cumpliendo con la lógica de la aplicación en los 3 casos.