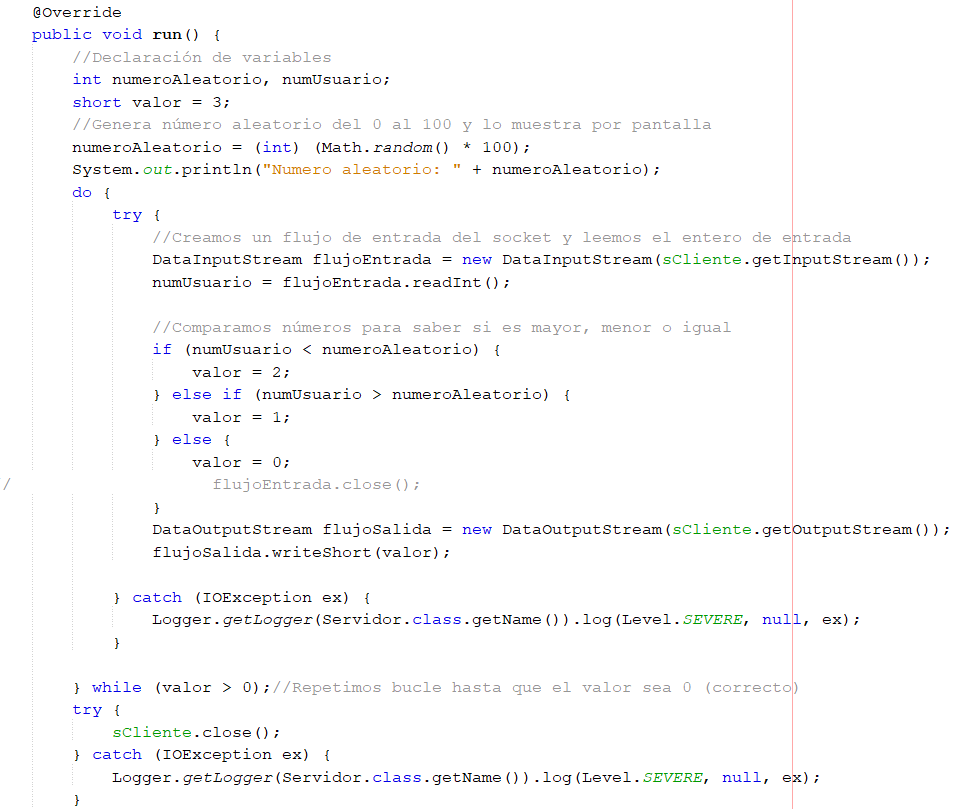
**ACTIVIDAD 4.1**

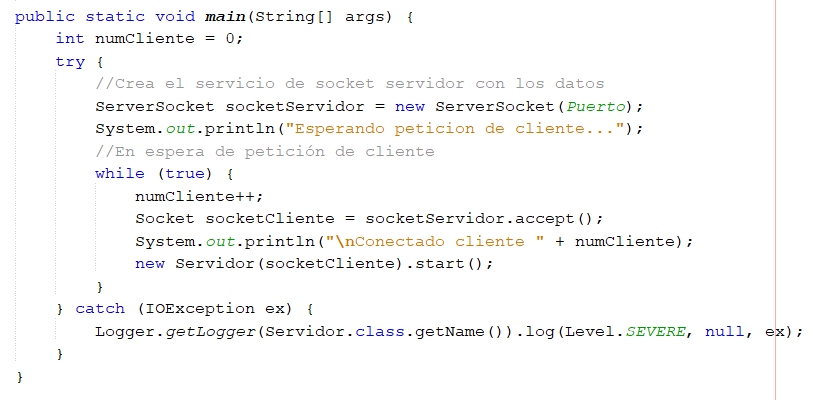
Modificamos la actividad 1 del tema 3 para hacer que el servidor trabaje con varias conexiones de clientes de manera concurrente. Para ello introduciré 2 modificaciones en el proyecto:

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



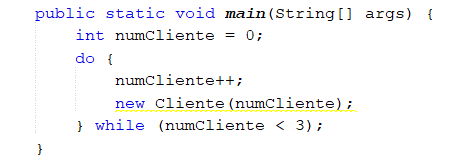
*Dentro del método vamos meter el código que realizará las operaciones de la clase*

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:



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

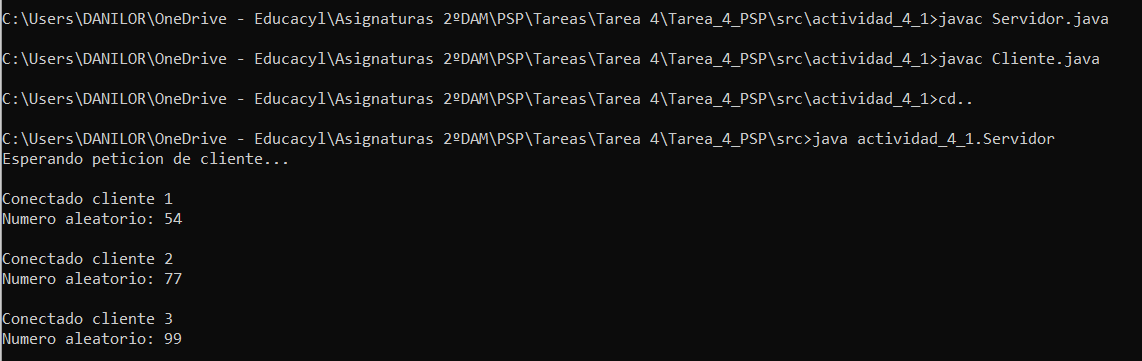
1. Dentro de la clase Cliente también vamos a modificar el método main() para hacer que se ejecute 3 veces la clase mediante un bucle, para probar la concurrencia de conexiones entre Cliente-Servidor:



*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:



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:



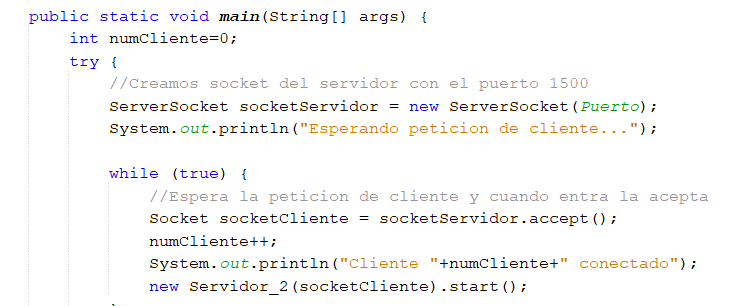
*Observamos que se conectan los 3 clientes de manera recurrente ejecutándose la aplicación completa para cada uno*

**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():

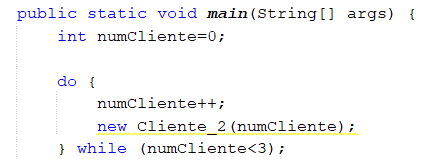






*Este último método va a contener el código de la lógica principal de la aplicación servidor*

1. 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:



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:

se cierra socket del cliente

False (máximo 3 intentos)

acepta

si no existe

(cierra socket cliente)

true

si existe

manda información al cliente

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.





*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:

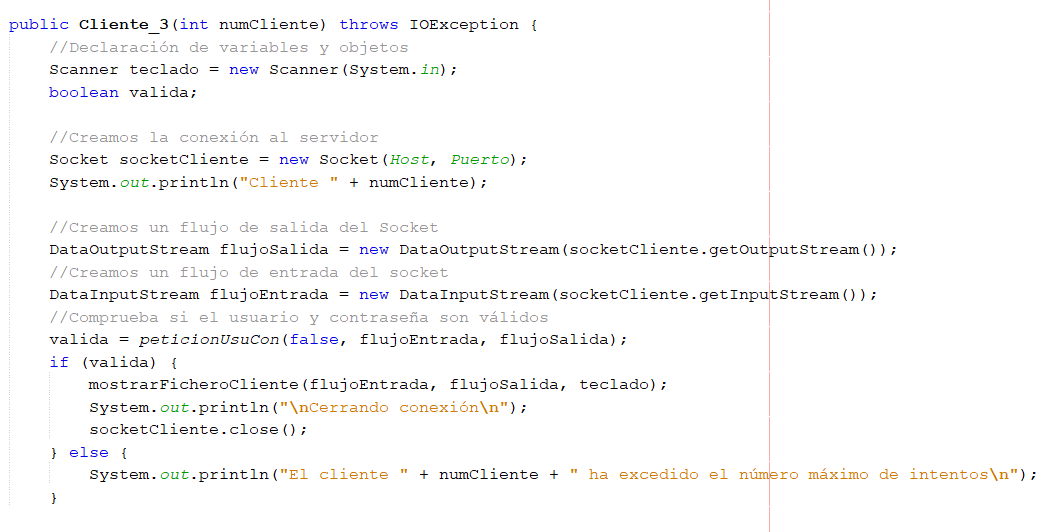








* *Cliente* dentro de esta clase definimos el constructor, el método main y creamos 2 métodos a mayores, como en la clase Servidor para poder organizar mejor el código:



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





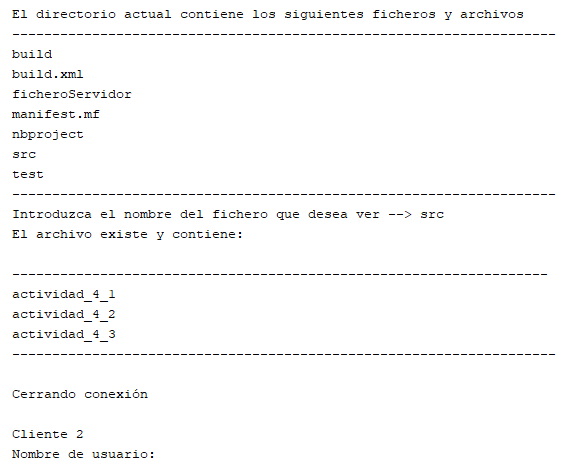
*Métodos para solicitar usuario y contraseña al usuario y método para mostrar por pantalla el fichero/archivo*

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:

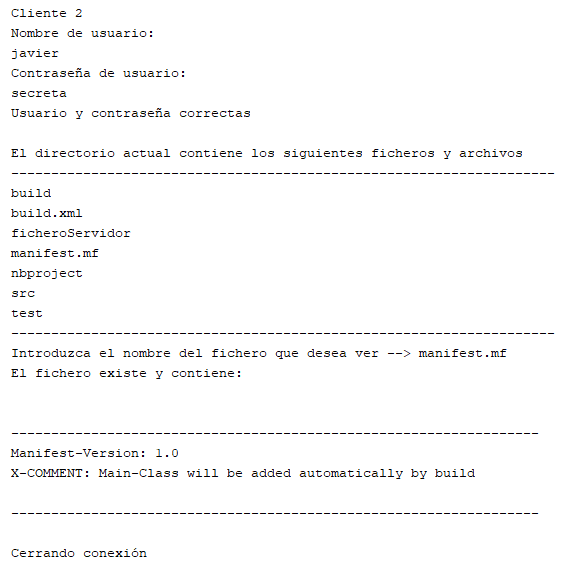
|  |  |
| --- | --- |
| **Cliente** | **Servidor** |
|  |  |
| Probamos la validación metiendo otro usuario y contraseña distinto a “javier”, “secreta” para a continuación mostrar contenido del proyecto | Mensaje de bienvenida y de información de tarea del servidor |

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



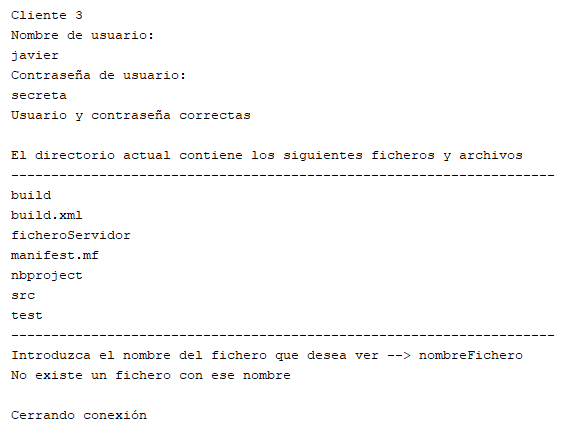
*Después de mostrarnos el contenido de la carpeta se cierra la conexión con el servidor del cliente número 1*

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



*Comprobamos que nos muestra por pantalla el contenido del fichero*

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



*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.