Programación de

Servicios y Procesos

CLIENTE-SERVIDOR

Diego González Martínez | 04/01/2023

# Ejercicio 4.2

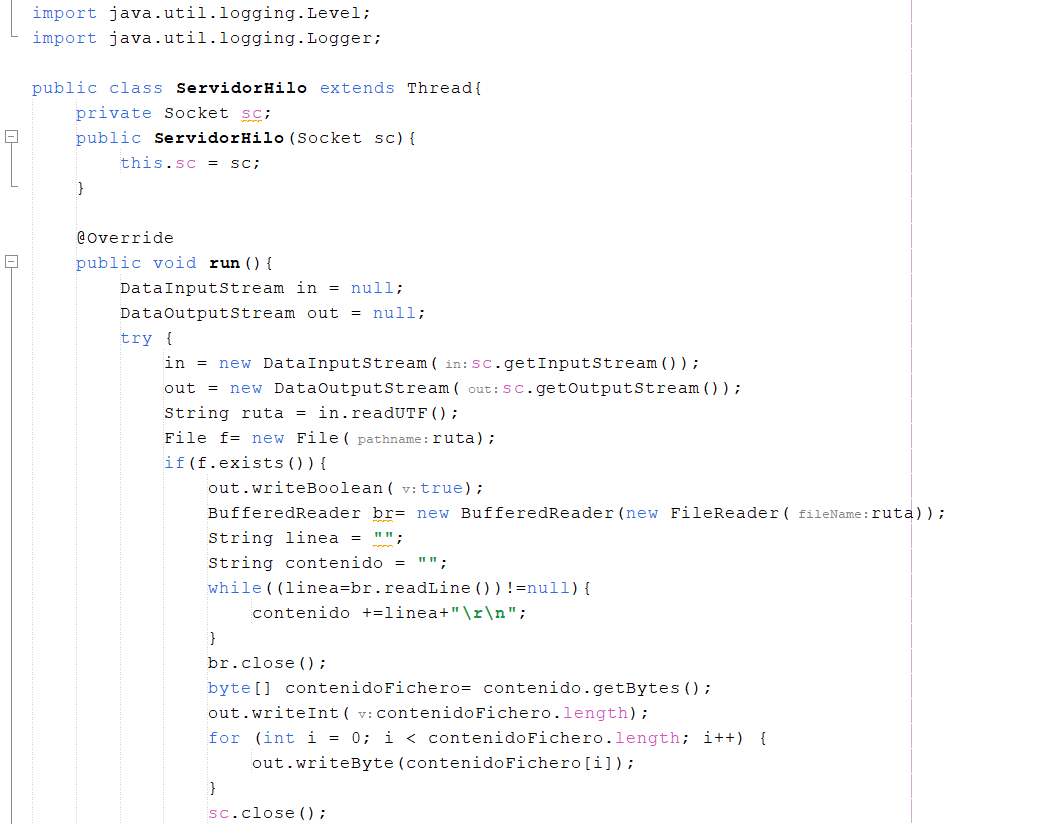
**¿De qué trata el Ejercicio 4?**

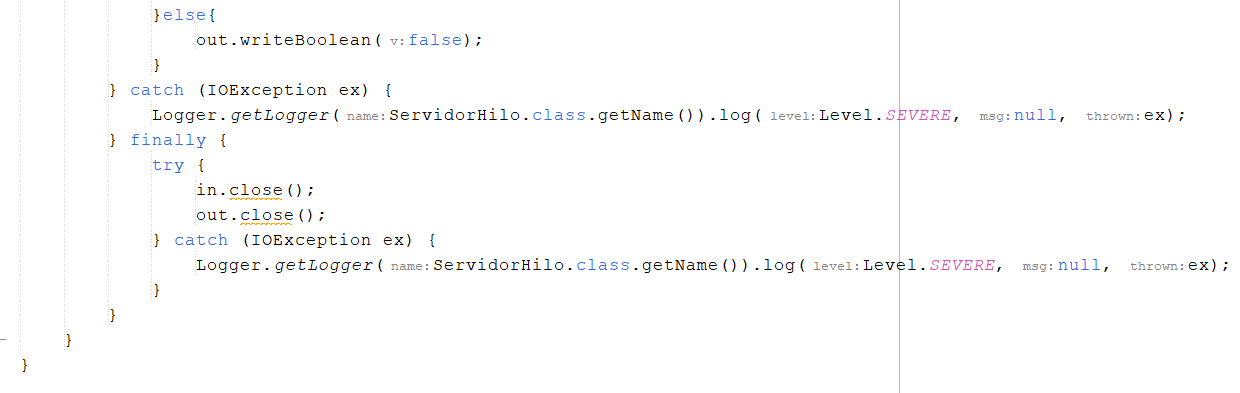
**Actividad 4.2.** Modifica el ejercicio 2 de la unidad 3 para el servidor permita trabajar de forma concurrente con varios clientes.

**La lógica del juego es el siguiente:**

1. El constructor de la clase recibe un socket y lo almacena en una variable miembro.
2. En el método "run", se crean dos streams de entrada y salida, uno para leer y otro para escribir.
3. Se recibe una ruta de un archivo a través del stream de entrada.
4. Se verifica si el archivo existe.
5. Si el archivo existe, se escribe un valor booleano "true" en el stream de salida, se lee el contenido del archivo y se escribe en el stream de salida el tamaño del contenido y el contenido en sí.
6. Si el archivo no existe, se escribe un valor booleano "false" en el stream de salida.
7. Finalmente, se cierran los streams y el socket.

**Clase Servidor-Hilo**

****

****

Este código es una clase de hilo del servidor que se utiliza para enviar un archivo desde el servidor al cliente.

La clase extiende la clase Thread e importa varios paquetes relacionados con la lectura y escritura de archivos y sockets.

En el constructor de la clase, se asigna el socket recibido al atributo de la clase.

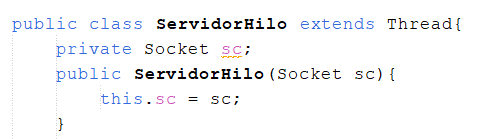
En el método "run", se crean dos objetos, "DataInputStream" y "DataOutputStream" utilizando los flujos de entrada y salida del socket. Se lee la ruta del archivo a enviar utilizando "readUTF" y se crea un objeto "File" con la ruta recibida.

Se verifica si el archivo existe. Si existe, se escribe un valor booleano verdadero en el flujo de salida y se lee el archivo línea por línea utilizando un "BufferedReader" y se guarda en una variable "contenido". El archivo se cierra y se convierte en un arreglo de bytes. Se escribe el tamaño del arreglo en el flujo de salida y se escribe cada byte del arreglo en el flujo de salida. Finalmente, se cierra el socket.

Si el archivo no existe, se escribe un valor booleano falso en el flujo de salida.

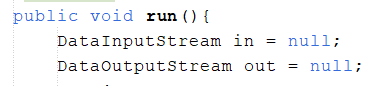
Finalmente, se cierran los flujos de entrada y salida en un bloque "finally" y se manejan las excepciones utilizando "try-catch" y "Logger".

**Explicación del código linea por linea:**

****

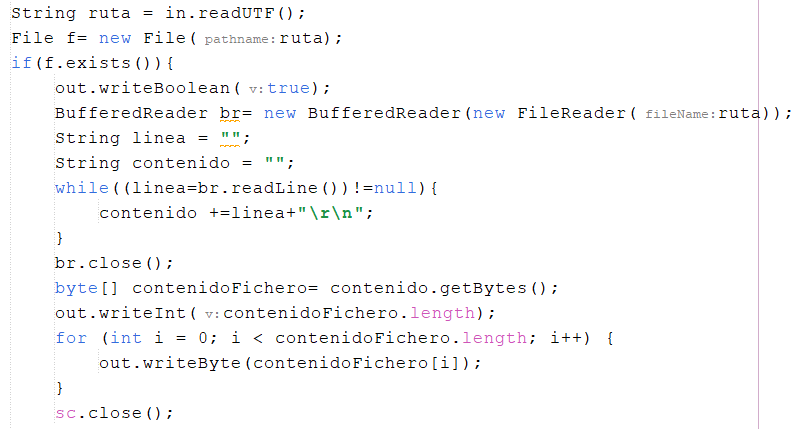
"private Socket sc;" declara un atributo privado de tipo Socket llamado "sc"

"public ServidorHilo(Socket sc){" inicializa el constructor de la clase y asigna el socket recibido al atributo "sc" de la clase.

****

"public void run() {" inicia el método "run", que es el método que se ejecuta cuando se inicia el hilo.

"DataInputStream in = null; DataOutputStream out = null; " declara e inicializa dos objetos, "DataInputStream" y "DataOutputStream", que se utilizarán para leer y escribir en el socket.

****

**"String ruta = in.readUTF();"** lee la ruta del archivo a enviar utilizando el método "readUTF" del objeto "in".

**"File f= new File(ruta);"** crea un objeto "File" con la ruta recibida.

**"if(f.exists()){"** comprueba si el archivo existe.

**"out.writeBoolean(true);"** si existe, escribe un valor booleano verdadero en el flujo de salida.

**"BufferedReader br= new BufferedReader(new FileReader(ruta));"** crea un **"BufferedReader"** para leer el archivo línea por línea.

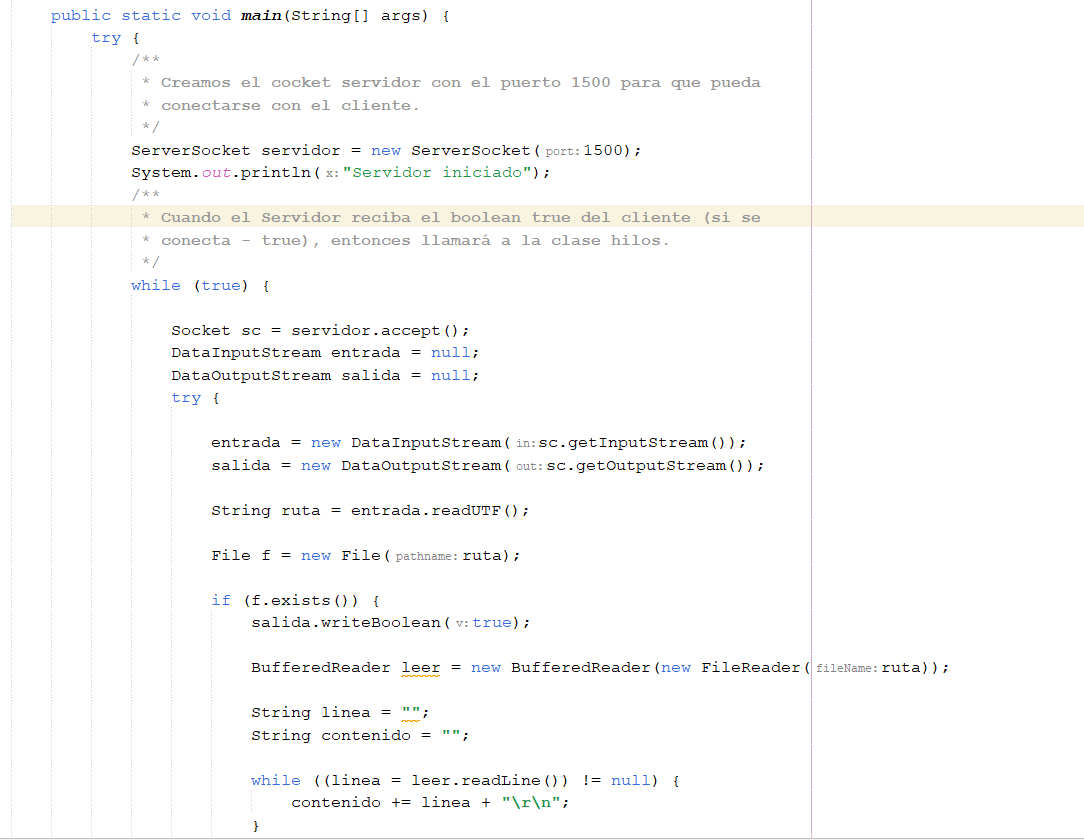
**"String linea = ""; String contenido = "";"** declara dos variables de tipo String vacías.

**"while((linea=br.readLine())!=null){"** comienza un ciclo while para leer cada línea del archivo.

**"contenido +=linea+"\r\n";"** agrega cada línea leída a la variable "contenido" y agrega un salto de línea.

**"br.close();"** cierra el " BufferedReader"

**Clase Servidor:**

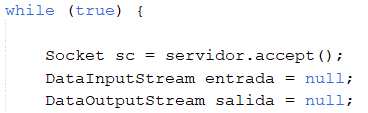


Creo un nuevo objeto ServerSocket llamado "servidor" en el puerto 1500. Esto permite al servidor recibir conexiones del cliente en el puerto especificado.



Creo un ciclo while infinito para aceptar nuevas conexiones. El servidor continuará aceptando conexiones mientras el programa esté en ejecución.

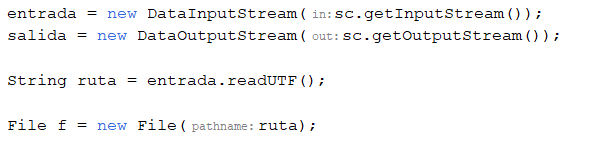
El método accept() del objeto servidor para esperar y aceptar una conexión del cliente. El método accept() es bloqueante, lo que significa que el programa se detendrá en esta línea hasta que se establezca una conexión.



Creo objetos DataInputStream y DataOutputStream para leer y escribir datos en la conexión. Estos objetos se utilizarán para enviar y recibir información entre el servidor y el cliente.

El método readUTF() del objeto entrada para leer la ruta del archivo enviada por el cliente. La ruta del archivo es una cadena que indica la ubicación del archivo en el sistema de archivos del servidor.

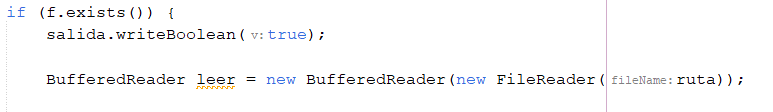
Creo un objeto File con la ruta del archivo. Este objeto se utilizará para verificar si el archivo existe en el sistema de archivos.



El método exists() del objeto File para verificar si el archivo existe. Si el archivo existe, el método devuelve true, de lo contrario devuelve false.

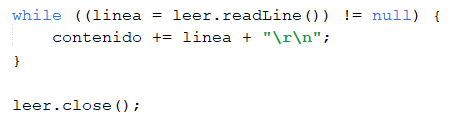
Si el archivo existe, utilizo el método writeBoolean() del objeto salida para enviar true al cliente y procede a leer el contenido del archivo.

Creo un objeto BufferedReader para leer el archivo. El objeto BufferedReader se utiliza para leer el archivo de manera más eficiente que utilizando el objeto FileReader directamente.



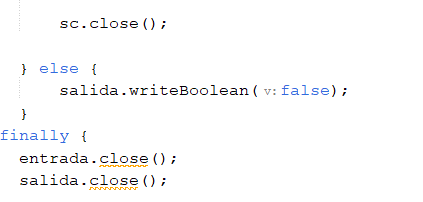
El método readLine() del objeto BufferedReader para leer cada línea del archivo y guardar en una variable String. El método readLine() devuelve null si se ha llegado al final del archivo.

El método close() del objeto BufferedReader para cerrar el archivo. Esto libera los recursos del sistema utilizados para acceder al archivo.



Si el archivo no existe, utilizo el método writeBoolean() del objeto salida para enviar false al cliente. Esto indica al cliente que el archivo no se encuentra en el servidor.

Finalmente, cierro los objetos entrada y salida utilizando el método close() de cada uno. Esto libera los recursos del sistema utilizados para acceder a la conexión.



Clase Cliente:

Este código Java es un programa cliente que se conecta a un servidor en el puerto 1500. El cliente solicita al usuario que ingrese una ruta de archivo, y la envía al servidor. El servidor luego verifica si el archivo existe, y si es así, envía el contenido del archivo de vuelta al cliente. El cliente luego muestra el contenido del archivo en la consola. Si el archivo no existe, el servidor envía un valor booleano de falso al cliente y el cliente muestra un mensaje de error. El programa utiliza las clases Socket, DataInputStream y DataOutputStream para la programación de sockets, y la clase Scanner para la entrada de usuario. También incluye un bloque try-catch para manejar cualquier excepción IOExceptions que pueda ocurrir.

Creo un nuevo objeto Socket llamado "sc" con el host "localhost" y el puerto 1500. Esto permite al cliente conectarse al servidor en ese host y puerto específico.

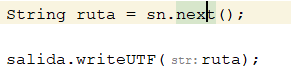


Creo objetos DataInputStream y DataOutputStream para leer y escribir datos en la conexión. Estos objetos se utilizarán para enviar y recibir información entre el cliente y el servidor.



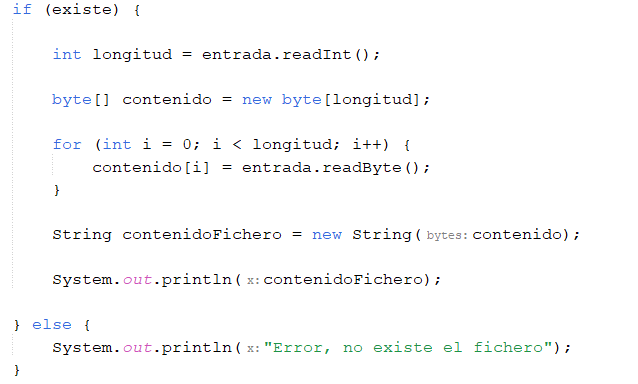
Utilizo el método next() del objeto Scanner para leer la entrada del usuario (la ruta del archivo).

Utilizo el método writeUTF() del objeto salida para enviar la ruta del archivo al servidor.

****

Si el archivo existe, utilizo el método readInt() para recibir la longitud del archivo, y un ciclo for para recibir el contenido del archivo byte por byte, si el archivo no existe, imprimo un mensaje de error indicando que el archivo no existe.

Finalmente, cierra la conexión utilizando el método close() del objeto sc. Esto libera los recursos del sistema utilizados para mantener la conexión.



**JAVADOC:**

C:\4.2Proyecto\target\site\apidocs

