Programación de

Servicios y Procesos

CLIENTE-SERVIDOR

Diego González Martínez | 30/12/2022

# Ejercicio 3.2

**¿De qué sirve el Ejercicio 3?**

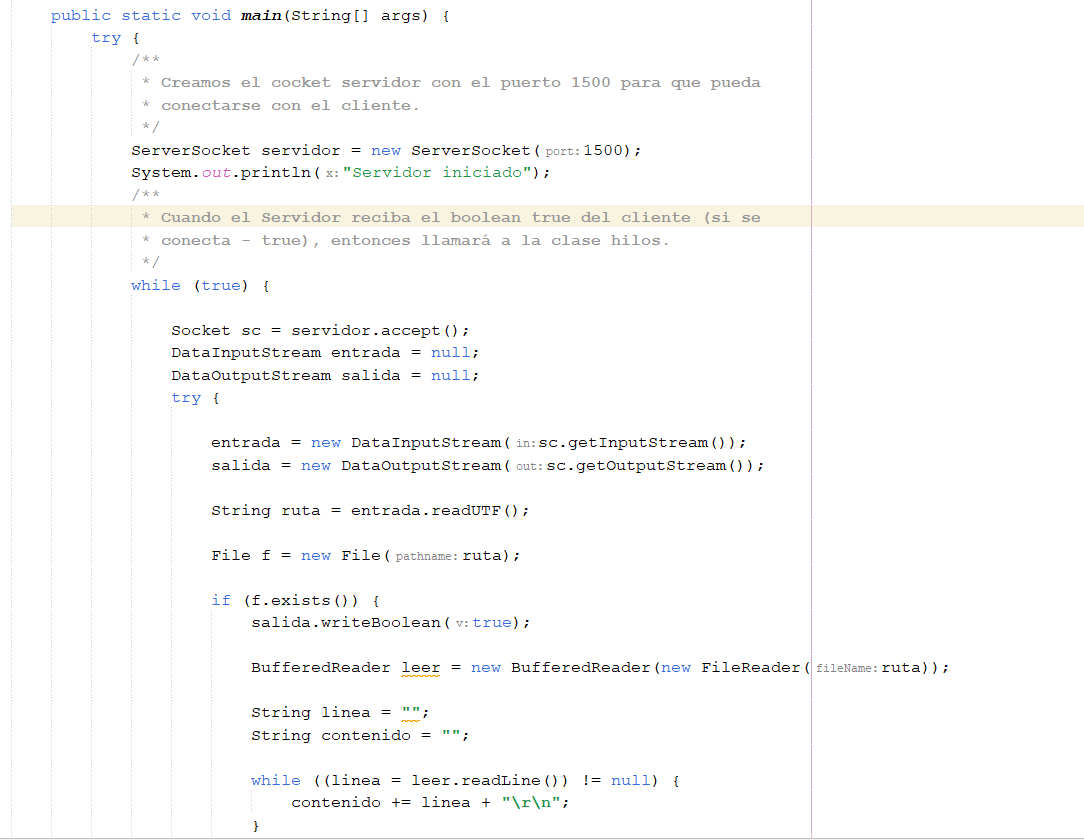
**Actividad 3.2**. El objetivo del ejercicio es crear una aplicación cliente/servidor que permita el envío de ficheros al cliente. Para ello, el cliente se conectará al servidor por el puerto 1500 y le solicitará el nombre de un fichero del servidor. Si el fichero existe, el servidor, le enviará el fichero 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.

Este código en Java es un programa de servidor que escucha las conexiones entrantes en el puerto 1500. Cuando se establece una conexión, lee una ruta de archivo del cliente, verifica si el archivo existe y, si es así, envía el contenido del archivo de vuelta al cliente. Si el archivo no existe, envía un valor booleano falso al cliente. Utiliza las clases ServerSocket, Socket, DataInputStream y DataOutputStream para la programación de sockets y la lectura de archivos. El programa también incluye un bloque try-catch para manejar cualquier excepción IOExceptions que pueda ocurrir.

**La lógica del juego:**

1. Creo un ServerSocket en el puerto 1500, que escucha las conexiones entrantes.
2. Una vez que se establece una conexión, el programa lee la ruta del archivo que es enviada por el cliente.
3. Luego, utilizo la clase File para verificar si el archivo existe.
4. Si el archivo existe, el programa utiliza un BufferedReader para leer el contenido del archivo y almacena el contenido en una variable de tipo string.
5. Luego, convierto la variable de tipo string a un arreglo de bytes y envía el arreglo de bytes al cliente utilizando el DataOutputStream.
6. Si el archivo no existe, envía un valor booleano falso al cliente.
7. El programa utiliza un ciclo while para mantener el servidor ejecutándose de forma continua y un bloque try-catch para manejar cualquier excepción de tipo IOException que pueda ocurrir.

**Clase Servidor:**

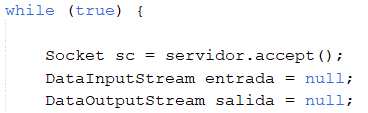


Creo un nuevo objeto ServerSocket llamado "servidor" en el puerto 1500. Esto permite al servidor recibir conexiones del cliente en el puerto especificado.



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

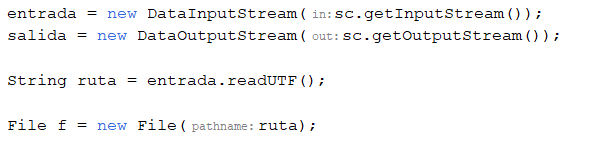
El método accept() del objeto servidor para esperar y aceptar una conexión del cliente. El método accept() es bloqueante, lo que significa que el programa se detendrá en esta línea hasta que se establezca una conexión.



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.

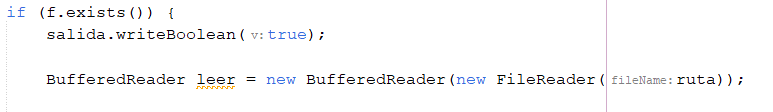
Creo un objeto File con la ruta del archivo. Este objeto se utilizará para verificar si el archivo existe en el sistema de archivos.



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.

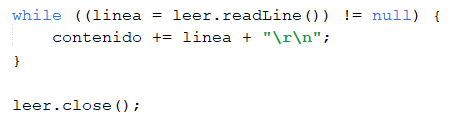
Si el archivo existe, utilizo el método writeBoolean() del objeto salida para enviar true al cliente y procede a leer el contenido del archivo.

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.



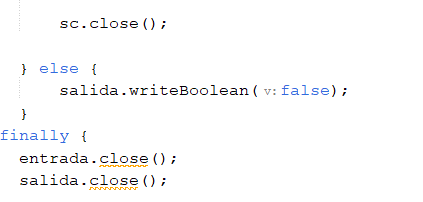
El método readLine() del objeto BufferedReader para leer cada línea del archivo y guardar en una variable String. El método readLine() devuelve null si se ha llegado al final del archivo.

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, cierra 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.

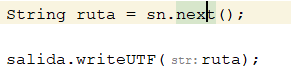
****

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.



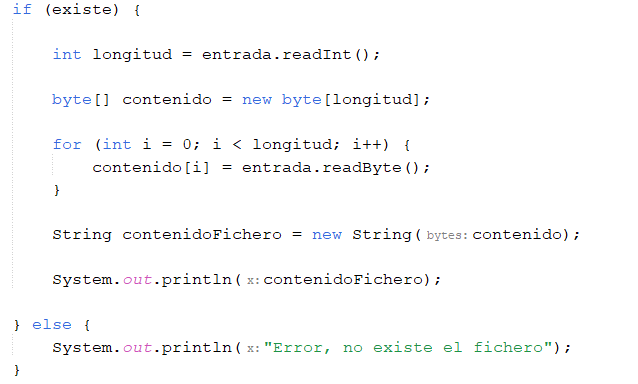
Utilizo el método next() del objeto Scanner para leer la entrada del usuario (la ruta del archivo).

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

****

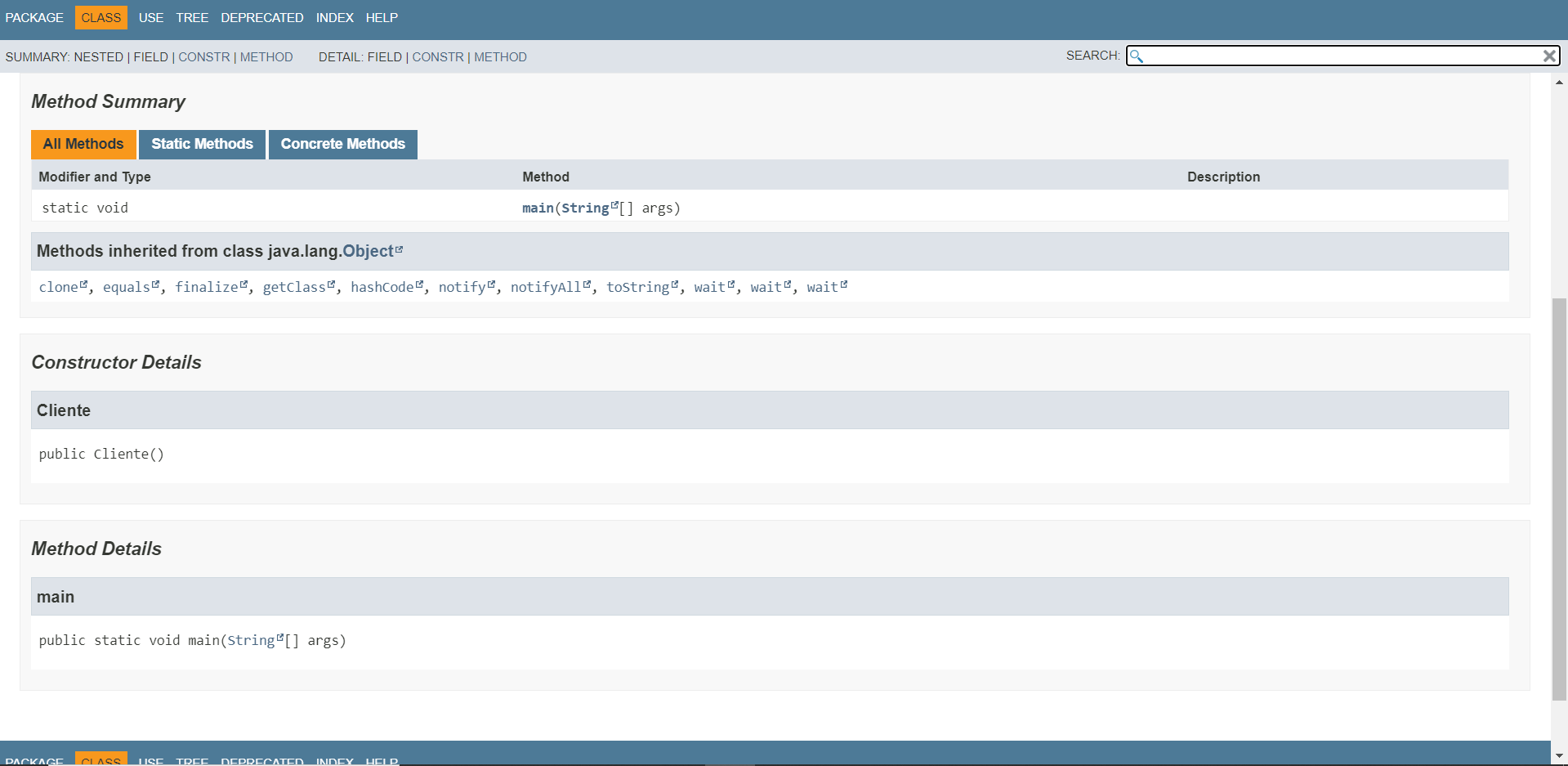
Si el archivo existe, se utiliza el método readInt() para recibir la longitud del archivo, y un ciclo for para recibir el contenido del archivo byte por byte, si el archivo no existe, se imprime un mensaje de error indicando que el archivo no existe.

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

****

JAVADOC:

**C:\3.2Proyecto\target\site\apidocs\**

****