Miguel Ángel Ramírez Pérez mramper0604@g.educaand.es

Explicación - Ejercicio 01 - Cliente-Servidor

Introducción

Este ejercicio pide hacer que dos clases cliente-servidor se comuniquen usando **Sockets** (***java.net.Socket*** y ***java.net.ServerSocket***).

El sistema funciona de la siguiente manera:

- El servidor escucha un puerto (54321).
- El cliente se conecta y envía mensajes o comandos (#).
- El servidor lee los comandos y envía respuestas.
- La comunicación termina cuando el cliente envía el comando ***#fin***.

Servidor

Gestiona la conexión del cliente y procesa los comandos que recibe.

¿Qué es un Socket y cómo funciona?

Un Socket es un mecanismo de comunicación que permite la transmisión de datos entre dos programas a través de la red:

- **ServerSocket(puerto):** Crea un servidor que escucha en un puerto específico
- **accept():** El servidor espera hasta que el cliente se conecte.
- **getInputStream():** Obtiene el flujo de entrada para recibir los datos que envía el cliente
- **getOutputStream():** Obtiene el flujo de salida para enviar los datos al cliente.

Ejemplo - Restaurante: Un restaurante tiene un sistema donde los camareros (clientes) envían pedidos a la cocina (servidor):

- **ServerSocket(54321):** La cocina está lista para recibir pedidos en el mostrador 54321.
- **Cliente se conecta -> accept():** Un camarero llega al mostrador para hacer un pedido.
- **Cliente envía comando -> readLine():** El camarero dice el pedido.
- **Servidor responde -> println():** La cocina confirma el pedido o da información.
- **Cliente envía #fin:** El camarero termina su turno y la cocina cierra la conexión.

Atributos

- **ServerSocket** *serverSocket***:** Socket principal que escucha las conexiones entrantes.
- **Socket** *clienteSocket***:** Socket específico para la comunicación con el cliente conectado.
- **PrintWriter** *printWriter***:** Flujo de salida para enviar mensajes al cliente.
- **BufferedReader** *reader***:** Ruta del directorio que el servidor utilizará para el comando ***#list***.
- **int** *PUERTO***:** Puerto de escucha del servidor (54321)

Métodos

- **start():** Inicia el servidor, solicita el directorio, espera la conexión y procesa los mensajes en un bucle hasta que reciba ***#fin***.

- **procesarComando(String comando):** Analiza el comando recibido y ejecuta la acción correspondiente:
    - **#fin:** Cierra la conexión.
    - **#info:** Envía información del host y puerto.
    - **#list:** Lista el contenido del directorio configurado.
    - Si no es reconocido, envía un mensaje de error.

- **listarFicheros(String ruta):** Recorre el directorio y envía la lista de archivos y carpetas al cliente, marcando con **F** (fichero) o **D** (directorio)

Ejecución:

1. **Solicita el directorio ->** Lee la ruta del directorio que envía el cliente-
2. **Crea ServerSocket ->** Inicia el servidor en el puerto 54321
3. **Espera al cliente ->** Se bloquea en ***accept()*** hasta que el cliente se conecte.
4. **Configura los flujos ->** Crea ***PrintWriter*** y ***BufferedReader*** para la comunicación.
5. **Bucle de mensajes ->** Lee los mensajes del cliente y los procesa
6. **Cierra los recursos ->** En el bloque ***finally*** garantiza que todos los recursos se cierran correctamente.
   El **finally** asegura que no queden recursos abiertos (sockets, streams).

Cliente

Representa a un cliente que se conecta al servidor para enviar mensajes y comandos. Cada ejecución del cliente establece una conexión independiente.

Atributos

- **Socket clienteSockets:** Socket para la conexión con el servidor.
- **PrintWriter printWriter:** Flujo de salida para enviar mensajes al servidor.
- **BufferedReader reader:** Flujo de entrada para recibir respuestas del servidor.
- **String HOST:** Dirección del servidor(***"localhost"***).
- **int PUERTO:** Puerto de conexión (***54321***).

Métodos

- **conectar():** Intenta establecer la conexión con el servidor. Crea el socket y configura los flujos de comunicación. Retorna ***true*** si la conexión fue exitosa, ***false*** en caso contrario.
- **enviarYRecibir():** Método principal que gestiona la interacción con el usuario:
    - Lee la entrada del usuario
    - Envía el mensaje al servidor
    - Recibe y muestra la respuesta completa
    - Finaliza al recibir el comando ***#fin***
- **leerRespuestaCompleta():** Lee todas las líneas de respuesta del servidor hasta encontrar el marcador FIN_RESPUESTA o el mensaje de cierre.
- **cerrar():** Cierra todos los recursos (PrintWriter, BufferedReader, Socket) de forma segura.

Miguel Ángel Ramírez Pérez mramper0604@g.educaand.es

Flujo de ejecución

1. **Conecta al servidor ->** Crea el socket y los flujos de comunicación.
2. **Solicita entrada ->** Espera que el usuario escriba un mensaje o comando.
3. **Envía al servidor ->** Transmite el mensaje mediante ***PrintWriter***.
4. **Lee respuesta ->** Recibe y muestra todas las líneas hasta el marcador de fin.
5. **Repite ->** Continúa el ciclo hasta que el usuario envíe ***#fin***.
6. **Cierra recursos ->** En el bloque ***finally*** garantiza que la conexión se cierra correctamente. El ***finally*** asegura que la desconexión se realiza incluso si ocurre un error.

Main

Servidor:

- Se ejecuta primero.
- Solicita al usuario la ruta del directorio a gestionar.
- Espera conexiones en el puerto 54321.
- Procesa comandos hasta recibir #fin.

Cliente:

- Se ejecuta después del servidor.
- Se conecta automáticamente a ***localhost:54321***.
- Permite al usuario enviar mensajes y comandos.
- Los comandos disponibles son:
    - **#info:** Muestra información del servidor.
    - **#list:** Lista el contenido del directorio configurado.
    - **#fin:** Finaliza la conexión.
- El cliente puede enviar mensajes de texto normales que el servidor simplemente confirma.

Datos importantes

- El servidor debe estar ejecutándose antes de iniciar el cliente.
- Solo un cliente puede conectarse a la vez.
- Todos los recursos se cierran automáticamente gracias a los bloques finally.