Instituto Politécnico Nacional

Escuela Superior de Computo

Paradigmas de Programacion

Practica 12

Profesor: García Floriano Andrés

Integrantes:

Elizalde Hernández Alan

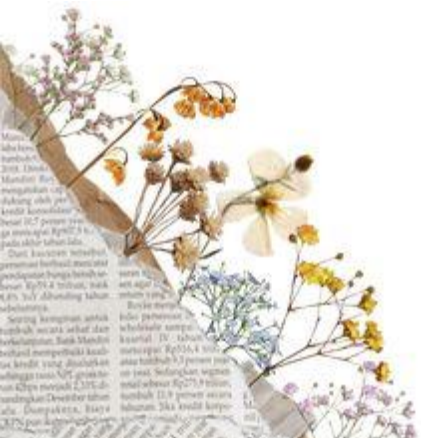
Reyes Ruiz Yoselyn Estefany

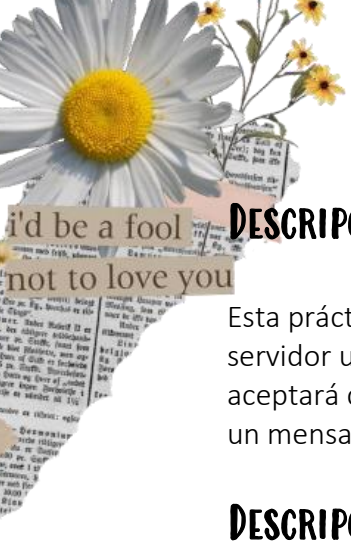
Solares Velasco Arturo Misael

Solís Lugo Mayra

Toral Hernández Leonardo Javier

3CV1





Descripción

Esta práctica tiene como objetivo implementar un sistema básico de comunicación cliente-servidor utilizando sockets en Java. El servidor escuchará en un puerto específico y aceptará conexiones de clientes, mientras que el cliente se conectará al servidor, enviará un mensaje y recibirá una respuesta.

Descripción de los Componentes

Servidor:

- Inicio del Servidor: El servidor se inicia y escucha en el puerto 5000.
- Aceptación de Conexiones: Utiliza ServerSocket para aceptar conexiones entrantes de clientes.
- Lectura de Datos del Cliente: Abre un stream para leer datos enviados por el cliente.
- Respuesta al Cliente: Envía una respuesta al cliente indicando su número de conexión.
- Cierre de Conexión: Cierra la conexión con el cliente después de enviar la respuesta.

Explicación del código

Clase principal y main

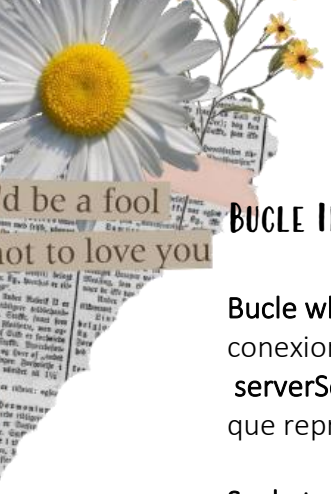
Clase Server: Define la clase principal del servidor.
Método main: Punto de entrada del programa. Aquí se inicia la ejecución del servidor.
Variable port: Define el puerto en el que el servidor escuchará las conexiones entrantes.

Variable numcliente: Lleva la cuenta del número de clientes que se han conectado

Creación del ServerSocket

ServerSocket: Clase que implementa un socket de servidor que espera a que los clientes se conecten.
new ServerSocket(port): Crea un ServerSocket que escucha en el puerto especificado.

try con recursos: Garantiza que el ServerSocket se cierre automáticamente al finalizar el bloque



Bucle Infinito para Aceptar Conexiones

Bucle while (true): Hace que el servidor esté siempre disponible para aceptar nuevas conexiones.
serverSocket.accept(): Espera y acepta una conexión de un cliente. Devuelve un Socket que representa la conexión con el cliente.

Socket clientSocket: Este Socket se usa para comunicarse con el cliente conectado.

Lectura de Datos desde el Cliente

BufferedReader in: Crea un BufferedReader para leer los datos enviados por el cliente a través del InputStream del Socket.
in.readLine(): Lee una línea de texto enviada por el cliente.

System.out.println("Mensaje del Cliente: " + inputLine): Imprime el mensaje del cliente en la consola del servidor

Envío de Respuesta al Cliente

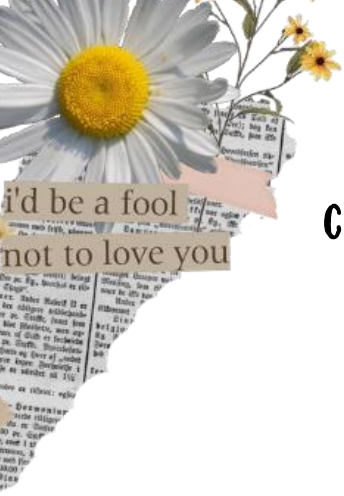
PrintWriter out: Crea un PrintWriter para enviar datos al cliente a través del OutputStream del Socket.
numcliente++: Incrementa el contador de clientes.

out.println(...): Envía un mensaje al cliente indicando su número de cliente.

Cierre de la Conexión con el Cliente

clientSocket.close(): Cierra la conexión con el cliente.

Manejo de Excepciones: Captura y maneja cualquier IOException que pueda ocurrir durante la aceptación de conexiones o la comunicación con el cliente



Cliente:

- **Conexión al Servidor:** El cliente se conecta al servidor en localhost en el puerto 5000.
- **Envío de Mensaje:** Envía un mensaje "Hola, servidor!" al servidor.
- **Recepción de Respuesta:** Lee la respuesta del servidor.
- **Manejo de Errores:** Gestiona errores de conexión y entrada/salida

Explicación del código

Clase principal y main

Clase Client: Define la clase principal del cliente.
Método main: Punto de entrada del programa. Aquí se inicia la ejecución del cliente.
Variable hostName: Define la dirección del servidor. En este caso, localhost indica que el servidor está en la misma máquina.
Variable port: Define el puerto en el que el servidor está escuchando

Creación del Socket y Conexión al Servidor

Socket socket: Crea un Socket que se conecta al servidor en la dirección y puerto especificados.
new Socket(hostName, port): Intenta establecer una conexión con el servidor.
PrintWriter out: Crea un PrintWriter para enviar datos al servidor a través del OutputStream del Socket.
out.println("Hola, servidor!"): Envía un mensaje al servidor.
System.out.println("Mensaje enviado al servidor."): Imprime un mensaje en la consola del cliente indicando que el mensaje fue enviado

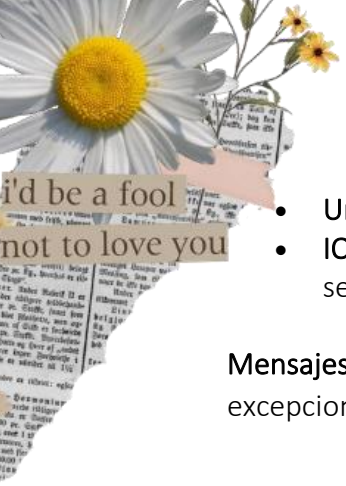
Lectura de Datos desde el Servidor

BufferedReader in: Crea un BufferedReader para leer los datos enviados por el servidor a través del InputStream del Socket.

in.readLine(): Lee una línea de texto enviada por el servidor.
System.out.println("Mensaje del Servidor: " + inputLine): Imprime el mensaje del servidor en la consola del cliente

Manejo de Excepciones y Cierre de Conexión

Manejo de Excepciones: Captura y maneja excepciones específicas:



- UnknownHostException: Ocurre si la dirección del servidor no se puede determinar.
- IOException: Ocurre si hay problemas de entrada/salida al intentar conectarse al servidor.

Mensajes de Error: Imprime mensajes de error específicos en la consola si ocurren excepciones

Resumen del Funcionamiento

Servidor:

- Inicia y escucha en el puerto 5000.
- Acepta conexiones de clientes de forma indefinida.
- Lee y muestra los mensajes enviados por los clientes.
- Envía una respuesta a cada cliente indicando su número de conexión.
- Cierra la conexión con cada cliente después de enviar la respuesta.

Cliente:

- Se conecta al servidor en localhost y puerto 5000.
- Envía un mensaje al servidor.
- Lee y muestra la respuesta del servidor.
- Maneja errores si la conexión falla o si hay problemas de entrada/salida

Conclusión

Esta práctica demuestra cómo implementar un simple sistema de comunicación cliente-servidor en Java utilizando sockets. El servidor es capaz de aceptar múltiples conexiones secuencialmente y responder a cada cliente con un mensaje personalizado. El cliente se conecta al servidor, envía un mensaje y espera una respuesta. La gestión de errores garantiza que el sistema maneje adecuadamente situaciones donde el servidor no esté disponible o haya problemas de conexión