

**Práctica: 3***Sockets Multicliente-Servidor*

Alumno: Reyes Reyes Jayim Javier

Profesor: Chadwick Carreto Arellano

Materia: Sistemas Distribuidos

Grupo:7CM1

Fecha de entrega: 6/03/25

# Antecedentes

Con el aumento de necesidades de eficiencia y rendimiento de un sistema y conexiones, surge el sistema multicliente con un servidor. A medida que los sistemas informáticos crecieron en complejidad, los servidores comenzaron a necesitar mecanismos para manejar múltiples clientes simultáneamente. En resumen, es la idea en que múltiples clientes pueden conectarse simultáneamente a un servidor centralizado para intercambiar información. Es utilizado en mensajería en tiempo real, juegos y muchas más aplicaciones.

Al igual que la arquitectura cliente-servidor, se establece la comunicación con sockets con los protocolos TCP o UDP. En este caso el servidor implementa hilos o threads para gestionar varias conexiones simultáneamente.

Por parte del cliente, se conecta para recibir y compartir información, en orden de realizar peticiones al servidor.

La concurrencia se maneja gracias a los hilos que permiten múltiples clientes de manera simultánea, cada cliente tiene su propio hilo de ejecución dentro del servidor.

Debido a la multiconexión es necesario que el servidor pueda manejar errores cuando los clientes se desconecten de forma inesperada.

Algunas alternativas de solución para los errores de conexión son implementar timeouts, reintentos y cierres adecuados de conexiones.

* Timeouts: Para detectar clientes inactivos.
* Mecanismos de reconexión: Para permitir a los clientes volver a conectarse.
* Logs y monitoreo: Para registrar actividad y depurar errores.

# Planteamiento del problema

Desarrollar un programa en Java que muestre un ejemplo de la conexión multicliente-servidor.

# Propuesta de solución

Para resolver el problema, se propone un modelo multicliente-servidor basado en sockets donde:

1. El servidor escucha en un puerto específico y acepta conexiones de múltiples clientes de forma concurrente.
2. Cada cliente envía un mensaje al servidor.
3. El servidor responde con un mensaje de confirmación al cliente.
4. El servidor maneja múltiples clientes usando hilos, asegurando que cada cliente sea atendido sin bloquear a los demás.

# Materiales y métodos empleados

Lenguaje: Java

Librerías: import java.io.\*;

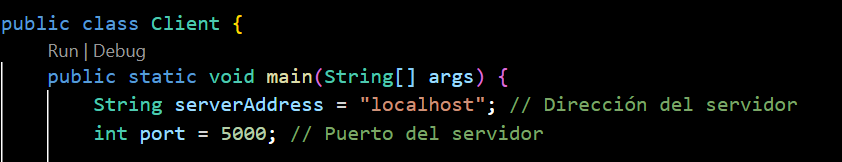
import java.net.\*;

import java.util.Scanner;

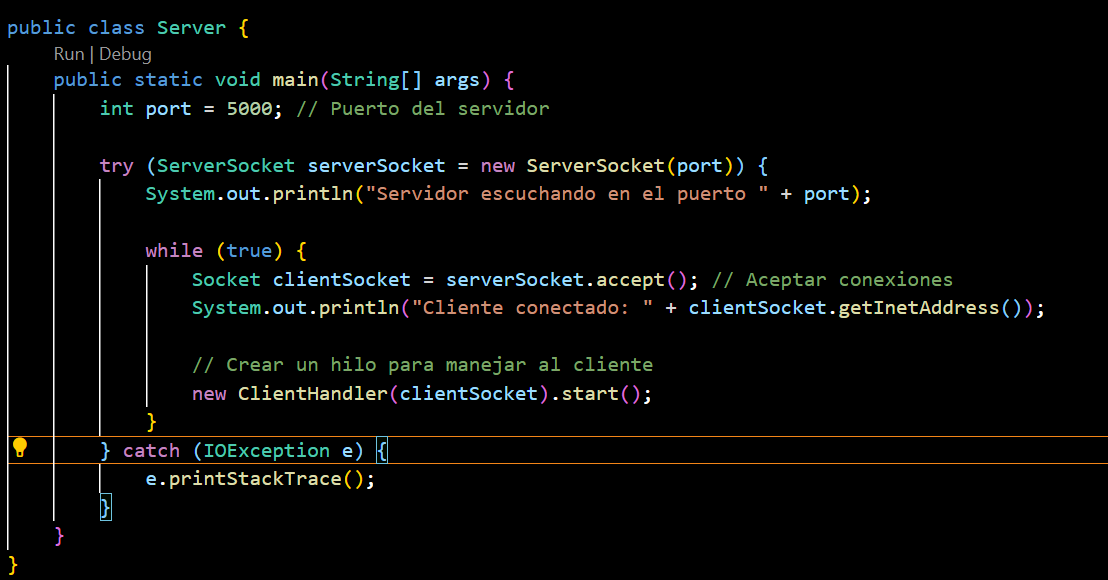
**LINK DE LA CARPETA ZIP:** [**https://github.com/Jayayim/Practica-3/archive/refs/heads/main.zip**](https://github.com/Jayayim/Practica-3/archive/refs/heads/main.zip)

**SE RETOMA EL CÓDIGO DE LA PRÁCTICA 2 CLIENTE-SERVIDOR Y SE ADAPTA A UN SISTEMA CON VARIOS HILOS.**

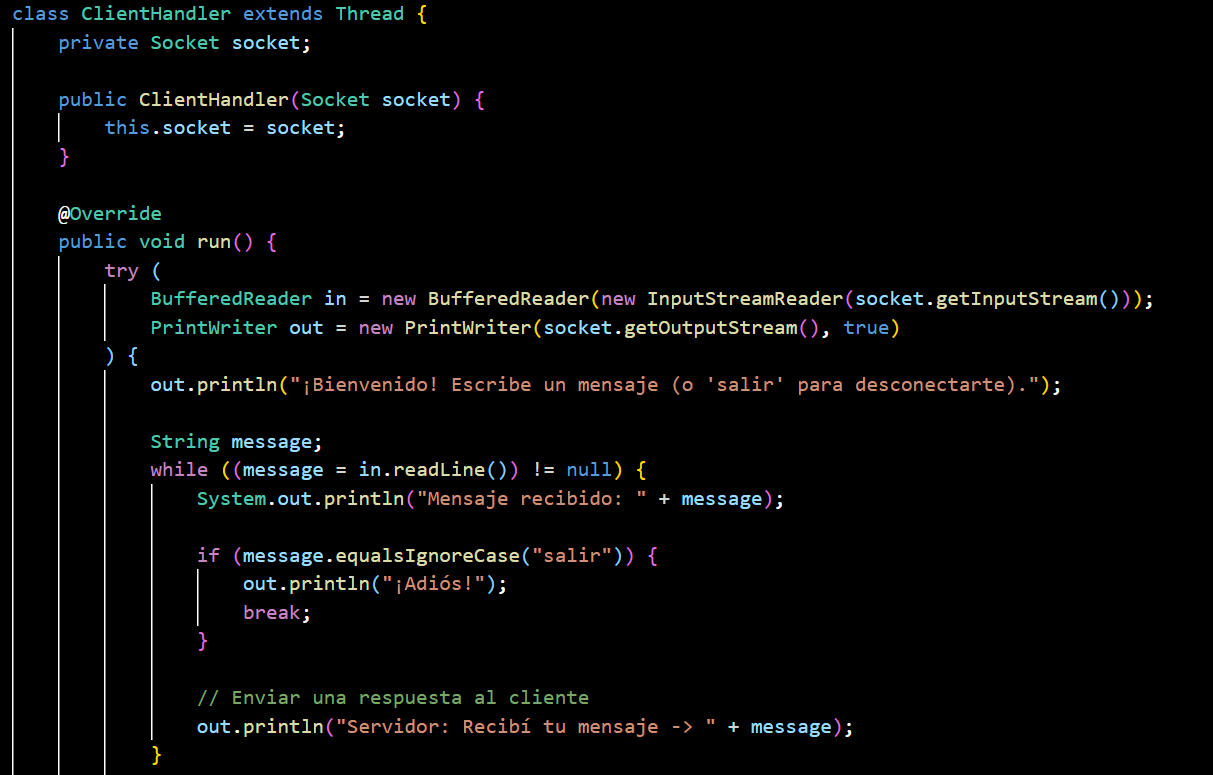
Código **SERVIDOR**



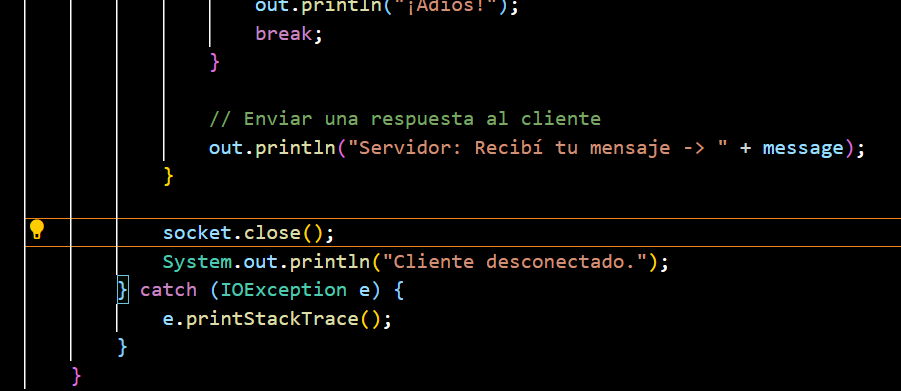
Primero definimos el puerto al que se conectarán el cliente y el servidor.



Dentro de un bloque Try Catch creamos el server como un socket y le pasamos el puerto, el cual será publicado para aceptar conexiones después, ya que utilizamos un modelo TCP. Ahí mismo creamos un clientsocket que se utiliza para iterar dentro de un while con el objetivo de aceptar las conexiones entrantes y después mostrar que se conectó un cliente con cierta dirección establecida. En este caso el programa es multihilo, acepta muchos clientes y es por ello el uso del ciclo y de la función ClientHandler para manejar todos los clientes.

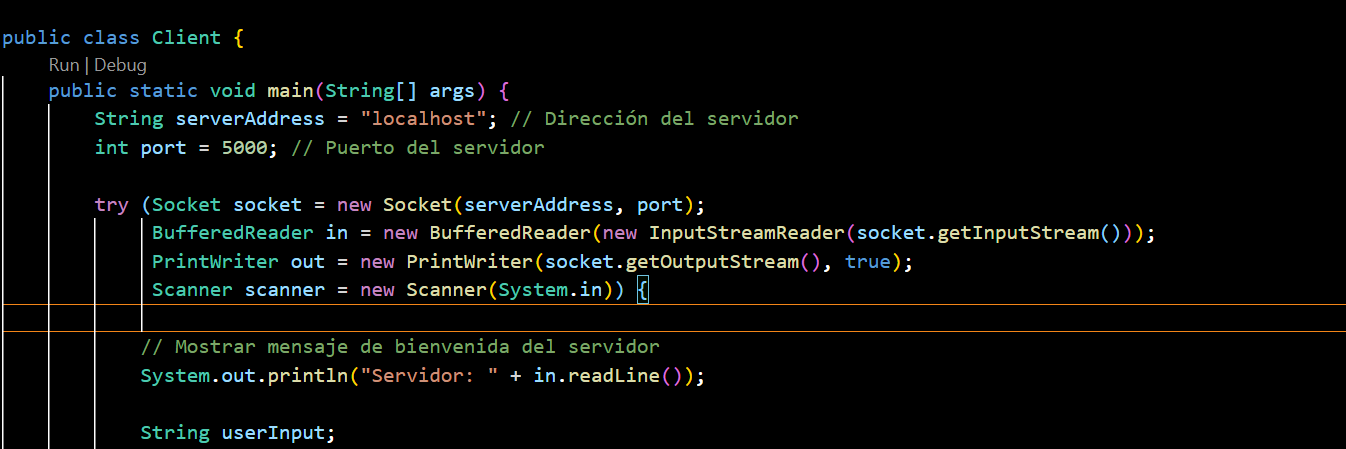


Primero utilizamos el manejo de hilos en clase ClientHandler por eso usamos el extends Thread. Después preparamos el buffer y el print que serán las formas en las que el server va a leer los mensajes recibidos por los clientes.

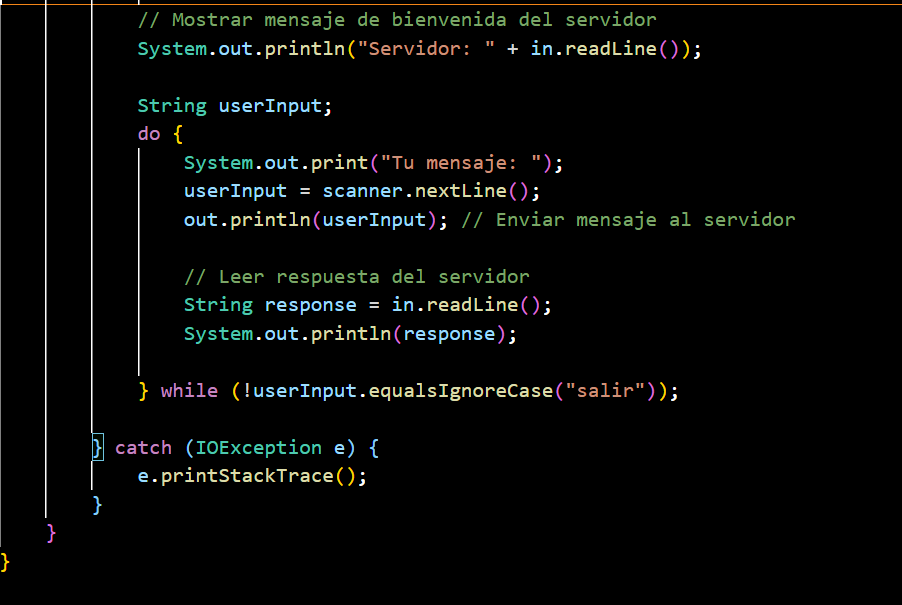


Importante es cerrar la conexión con los hilos.

Codigo **CLIENTE**

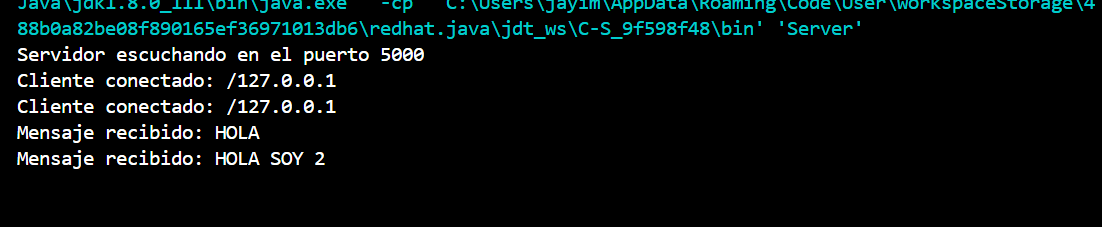


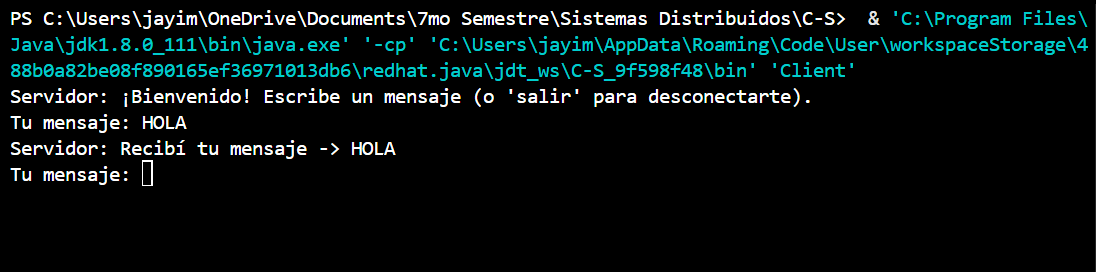
Inicializamos el puerto y el socket de la misma manera que en el servidor. Creamos el buffer y el print para mandar los datos al servidor y recibirlos.

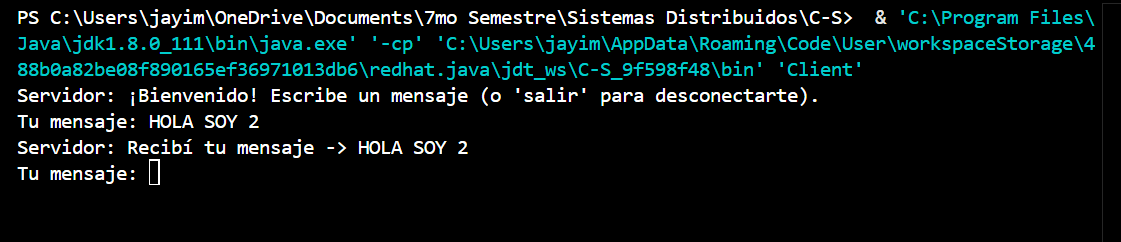


Finalmente, solo mandamos el mensaje al servidor y leemos su respuesta.

# Resultados







# Conclusiones

Si se manejan de forma correcta, los sistemas multicliente con un solo servidor son bastantes útiles ya que permiten un buen manejo de concurrencia, protocolos de comunicación y construir aplicaciones escalables, sin embargo, todo dentro de un entorno moderado, si se sobrecarga por n numero de clientes o si se busca escalar a una red mucho más grande, las características de este modelo limitan el uso de recursos para estos fines.