

**Práctica: 2***Sockets Cliente-Servidor*

Alumno: Reyes Reyes Jayim Javier

Profesor: Chadwick Carreto Arellano

Materia: Sistemas Distribuidos

Grupo:7CM1

Fecha de entrega: 3/03/25

# Antecedente

Después de que las necesidades mejorar la velocidad de procesamiento y de tener un mejor control en los recursos empezaban a satisfacerse con la implementación de Sistemas Distribuidos, en la década de 1970 a 1980 se comenzó a descentralizar mucho más el procesamiento, empezaron a surgir los primeros modelos cliente-servidor, donde el servidor era el que brindaba servicios a uno o varios clientes.

El modelo cliente-servidor es muy utilizado en varias aplicaciones de red, como BD, juegos, mensajes, páginas, etc.

Para profundizar en esta práctica es importante puntualizar algunos conceptos:

**Cliente:** Un cliente es un programa o dispositivo que manda solicitudes a un servidor y queda en espera de una respuesta.

**Servidor:** Un servidor es un programa que recibe peticiones o escucha las solicitudes de clientes, las procesa y manda una respuesta. Es capaz de manejar múltiples clientes simultáneamente.

**Protocolo de Comunicación:** Son las reglas para establecer la conexión de cliente y servidor.

**Socket:** Punto de comunicación entre programas donde se encuentra la emisión y recepción de datos.

Existen dos tipos de sockets.

**TCP:**

* Usan el protocolo TCP (Transmission Control Protocol).
* Son confiables, garantizan que los datos lleguen sin errores y en orden.

**UDP:**

* Usan el protocolo UDP (User Datagram Protocol).
* Son más rápidos, pero no garantizan que los datos lleguen correctamente o en orden.

Los pasos esenciales para poder realizar una conexión de Cliente-Servidor:

SERVIDOR:

* Crear un socket usando socket.socket().
* Vincularlo a una dirección IP y un puerto con bind().
* Escuchar conexiones entrantes usando listen().
* Aceptar una conexión de un cliente con accept().
* Recibir datos del cliente con recv().
* Procesar la solicitud y enviar una respuesta con send().
* Cerrar la conexión cuando termine la comunicación.

CLIENTE:

* Crear un socket usando socket.socket().
* Conectarse al servidor con connect().
* Enviar una solicitud con send().
* Esperar y recibir la respuesta con recv().
* Procesar la respuesta.
* Cerrar la conexión.

# Planteamiento del problema

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

# Propuesta de solución

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

1. El servidor escucha en un puerto específico y acepta conexión a un cliente.
2. Cada cliente envía su dirección al server.
3. El servidor muestra un mensaje de confirmación al cliente y su dirección.

# Materiales y métodos empleados

Lenguaje: Java

Librerías: import java.io.\*;

import java.net.\*;

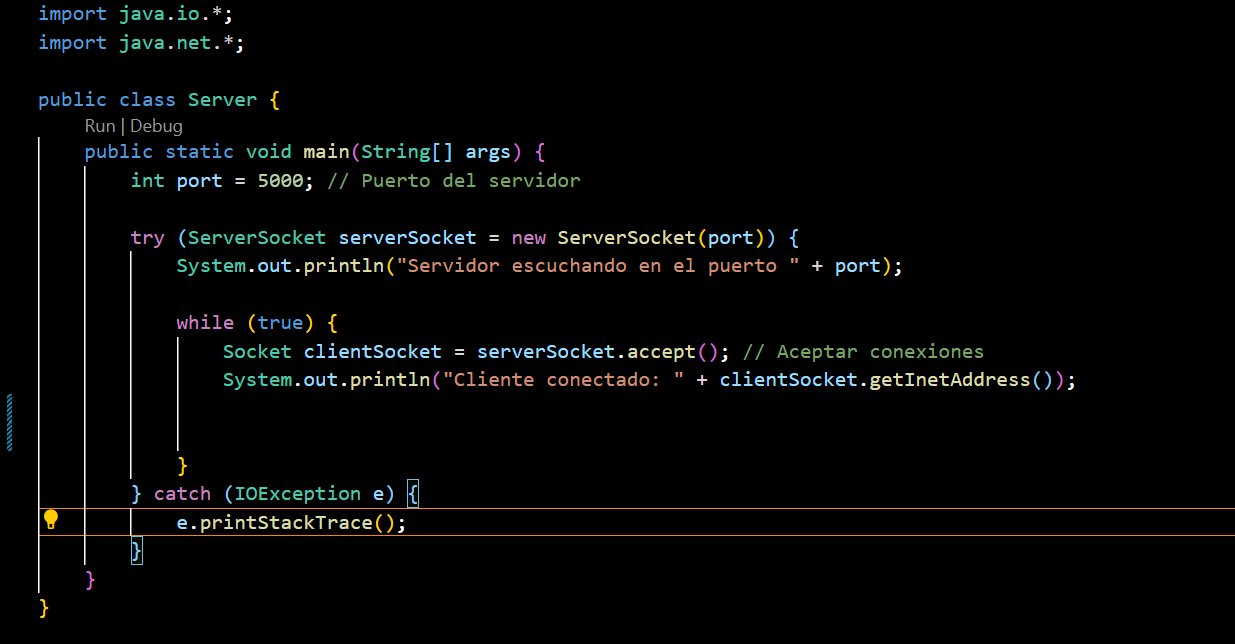
import java.util.Scanner;

**LINK DE LA CARPETA ZIP DESCARGA AUTOMÁTICA:**

[**https://github.com/Jayayim/Practica-2/archive/refs/heads/main.zip**](https://github.com/Jayayim/Practica-2/archive/refs/heads/main.zip)

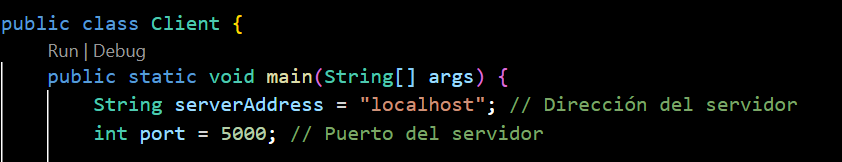
Código **SERVIDOR**

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

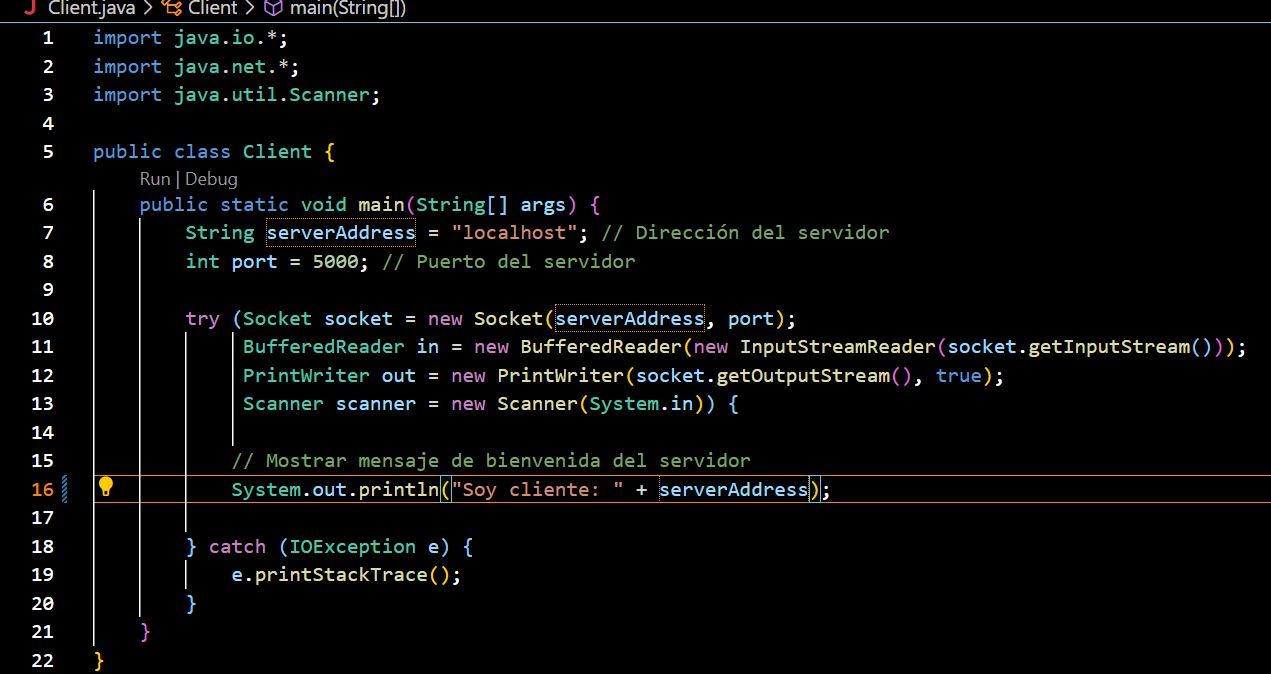


Dentro de un bloque Try Catch creamos el server como un socket y le pasamos el puerto, el cual será publicado para aceptar conexiones después, ya que utilizamos un modelo TCP.

Codigo **CLIENTE**

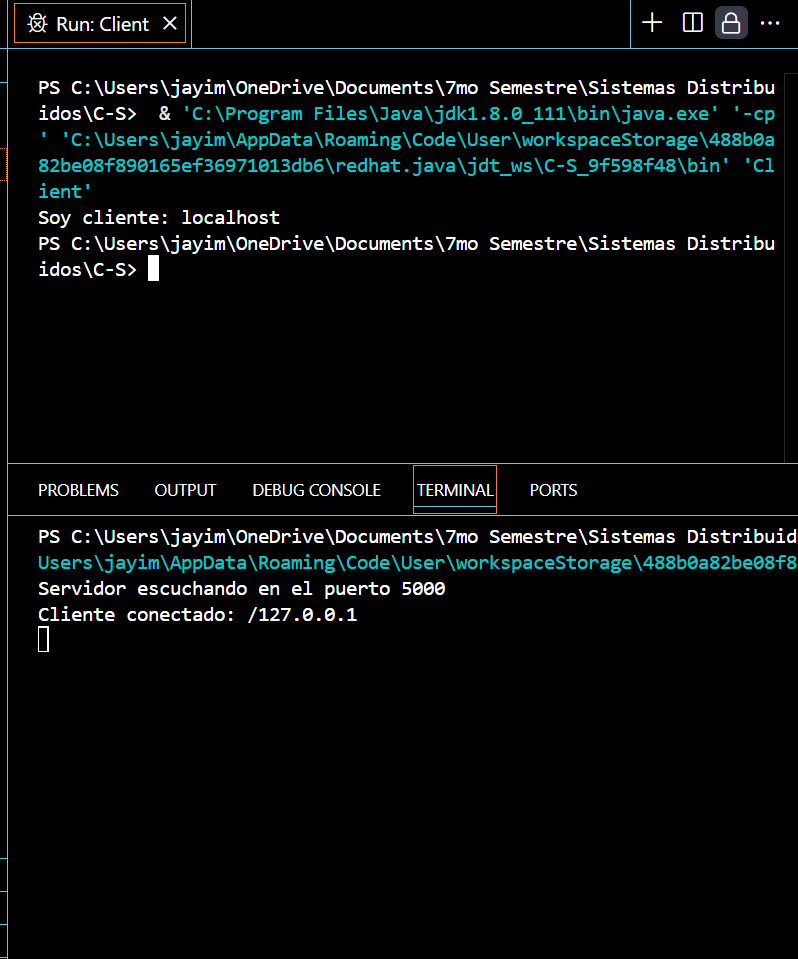


Definimos el puerto para conectar.



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.

# Resultados



# Conclusiones

El esquema de cliente-servidor es importante para casi toda la lógica de sistemas distribuidos, el servidor permite gestionar las solicitudes y brindar respuestas a varios clientes, también se puede escalar con la finalidad de compartir recursos, administrar conexiones concurrentes y garantizar la comunicación según se requiera. Existen algunas áreas de oportunidad como el control cuando sean servidores y clientes masivos, sin embargo, en esta práctica sencilla se pueden manejar de manera eficiente.