

**INSTITUTO TECNOLOGICO DE CHILPANCINGO**

**Cómputo En La Nube Y Grandes Datos**

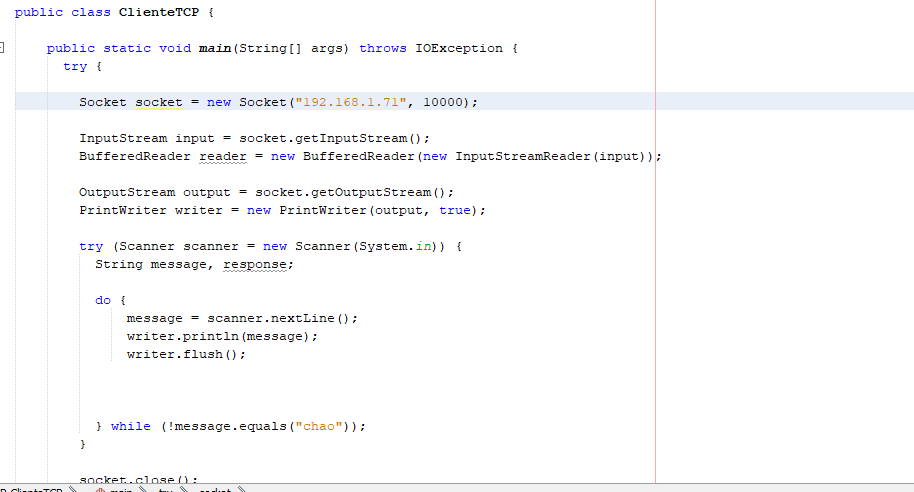
Practica Sockets TCP y UDP Cooperativo Java

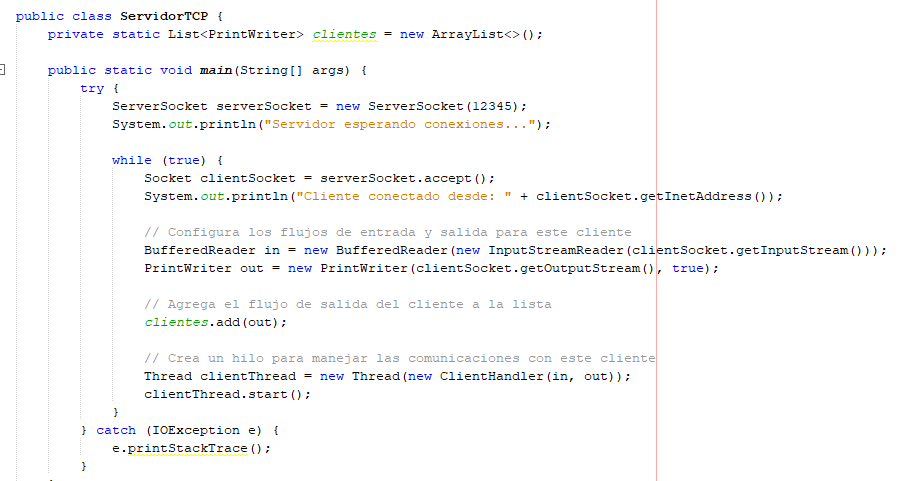
**Alumno:**

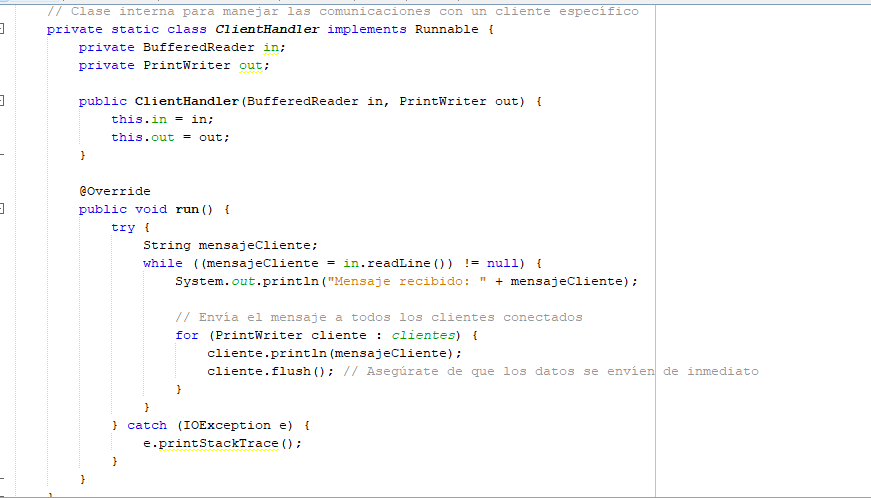
Fierro Monroy Isaac Antonio

Lunes 13 de mayo de 2024

# Sockets TCP







Se utiliza un Socket para establecer la conexión con el servidor en la dirección IP "192.168.1.71" en el puerto 10000.

Se obtienen los flujos de entrada y salida del socket (InputStream y OutputStream) para enviar y recibir datos al y desde el servidor.

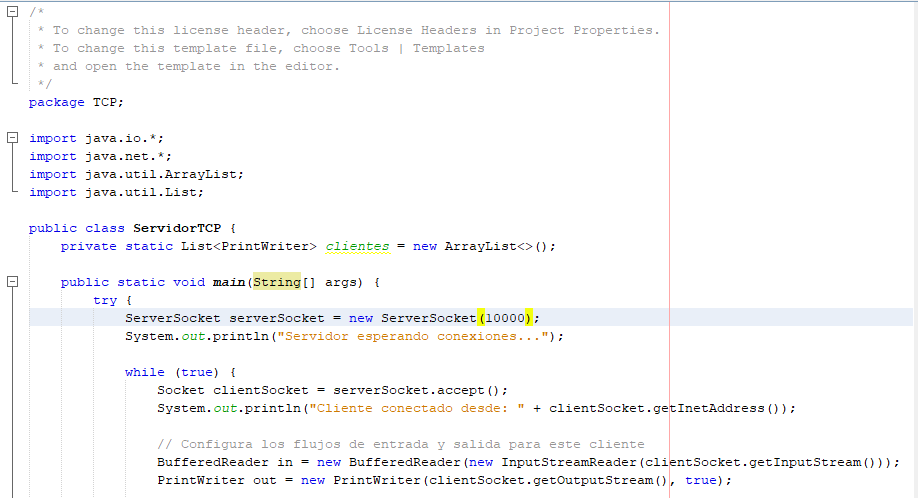
Se utiliza un BufferedReader para leer las respuestas del servidor.

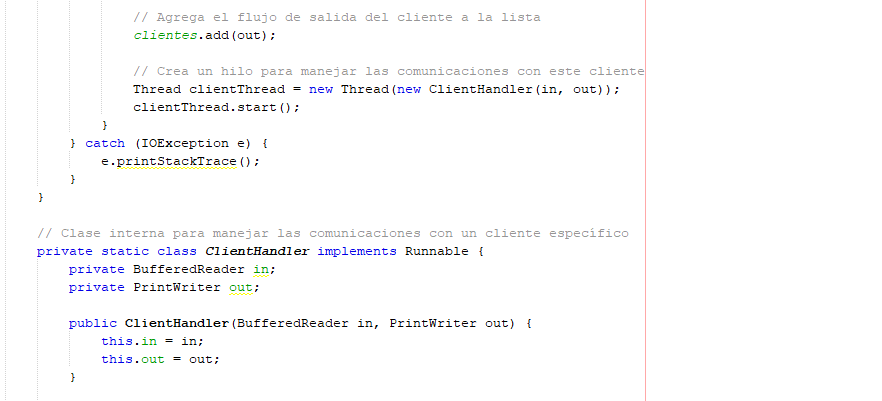
Se utiliza un PrintWriter para enviar mensajes al servidor.

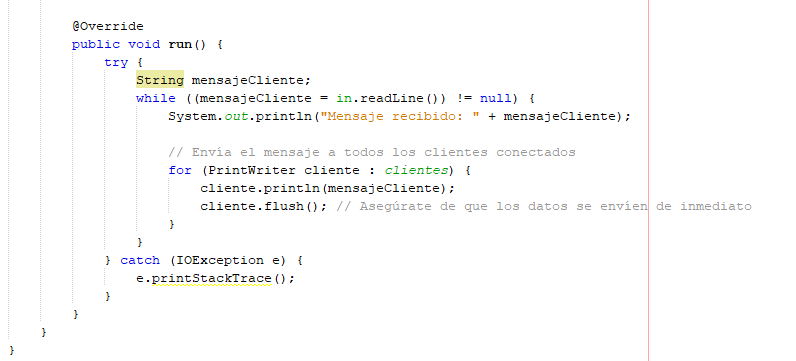
Se utiliza un Scanner para leer la entrada del usuario desde la consola.

El cliente espera que el usuario ingrese mensajes desde la consola y los envía al servidor. El bucle do-while continúa hasta que el usuario ingresa "chao".

Una vez que se envía "chao", se cierra la conexión con el servidor.







Se crea un ServerSocket que escucha en el puerto 10000 para nuevas conexiones entrantes.

Dentro de un bucle infinito, el servidor acepta conexiones entrantes utilizando el método accept() del ServerSocket.

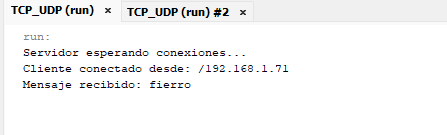
Para cada cliente que se conecta, se crea un nuevo hilo (ClientHandler) para manejar las comunicaciones con ese cliente.

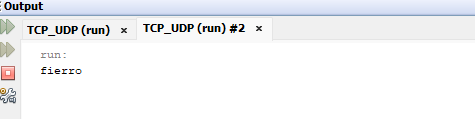
En el ClientHandler, se configuran flujos de entrada y salida para leer mensajes del cliente y enviar mensajes a todos los clientes conectados.

El servidor espera mensajes del cliente en un bucle while, leyendo líneas de texto del flujo de entrada del cliente.

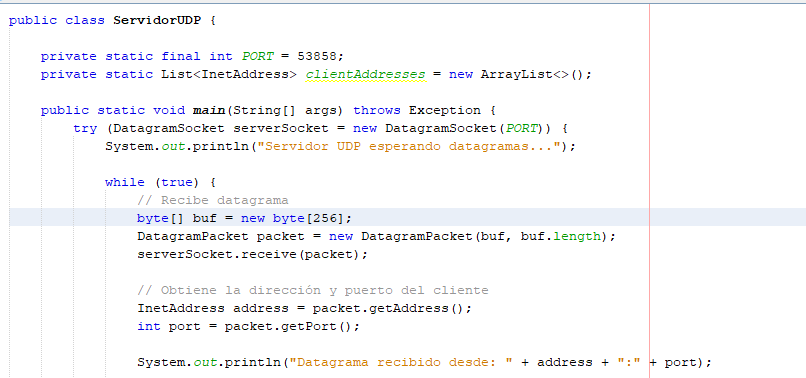
Cuando se recibe un mensaje del cliente, se imprime en la consola del servidor y se envía a todos los clientes conectados a través de sus flujos de salida.

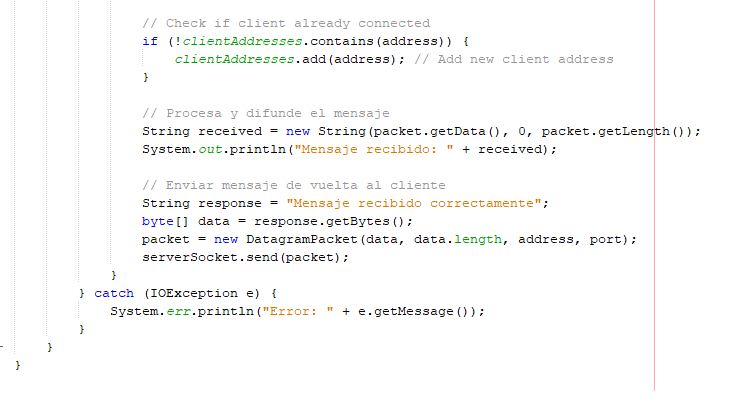
Si ocurre un error durante la lectura de mensajes del cliente, se imprime una traza de la pila del error.





# Sockets UDP





Se crea un socket UDP (DatagramSocket) en el puerto especificado (53858).

Dentro de un bucle infinito, el servidor espera la llegada de datagramas.

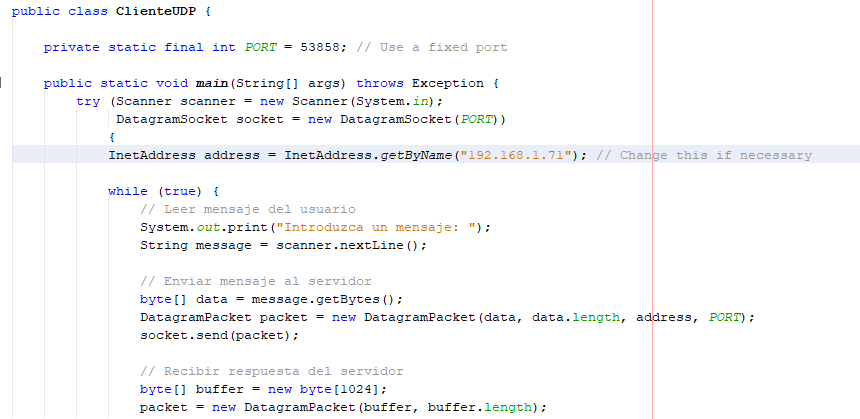
Cuando llega un datagrama, se extraen los datos y la información sobre la dirección y el puerto del remitente del paquete (DatagramPacket).

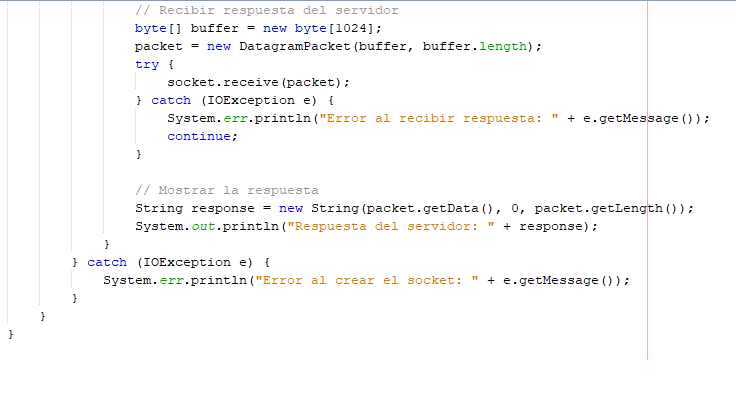
Si es la primera vez que se recibe un datagrama del cliente (identificado por su dirección IP), se agrega su dirección IP a la lista de direcciones de clientes.

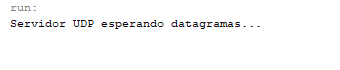
El mensaje recibido se imprime en la consola del servidor.

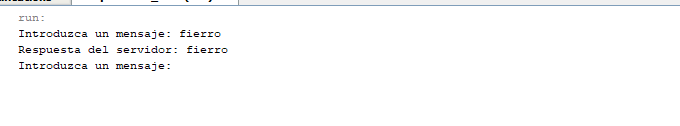
Se prepara un mensaje de respuesta ("Mensaje recibido correctamente") y se envía de vuelta al cliente que envió el datagrama.

El servidor continúa escuchando para más datagramas entrantes.









Se crea un socket UDP (DatagramSocket) que utilizará para enviar y recibir datagramas.

Se utiliza un bucle infinito para permitir al usuario enviar múltiples mensajes al servidor.

El cliente espera que el usuario ingrese un mensaje desde la consola.

El mensaje del usuario se convierte en un arreglo de bytes (byte[]) y se envía al servidor como un datagrama UDP utilizando un objeto DatagramPacket.

El cliente espera recibir una respuesta del servidor. Se configura un DatagramPacket para recibir la respuesta y se utiliza el método receive() del DatagramSocket para esperar a que llegue la respuesta del servidor.

Una vez que se recibe la respuesta del servidor, se convierte de bytes a una cadena de texto y se muestra en la consola del cliente.

# CONCLUSION

Los fragmentos de código presentados son ejemplos simples de comunicación utilizando sockets en Java, específicamente implementando la comunicación mediante el protocolo TCP/IP.

* Servidor TCP/IP: Este fragmento de código crea un servidor que espera conexiones entrantes de clientes. Una vez que se establece una conexión, el servidor puede recibir mensajes del cliente, procesarlos y enviar respuestas de vuelta al cliente.
* Cliente TCP/IP: Por otro lado, el fragmento de código del cliente se conecta al servidor y permite al usuario enviar mensajes al servidor. Después de enviar un mensaje, el cliente espera recibir una respuesta del servidor, que luego muestra en la consola.

Ambos fragmentos de código están diseñados para operar de forma continua, lo que significa que el servidor permanece a la espera de conexiones y el cliente permanece interactivo, permitiendo al usuario enviar mensajes al servidor y recibir respuestas.