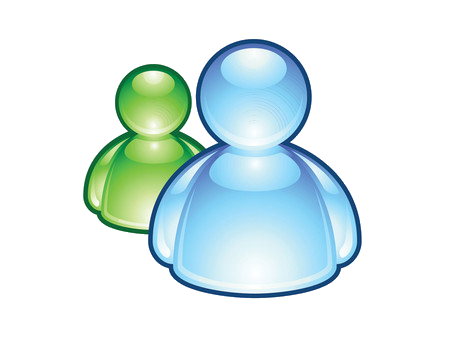


***Instituto Politecnico Nacional***

**Escuela Superior de Cómputo**

**Bocanegra Heziquio Yestlanezi   
  Practica 5  
  Sockets Clientes   
1. Chat 3D   
Fecha: 11 de junio de 2021   
  2CM13  
  Programación Orientada a Objetos**



Contenido

[introducción 3](#_Toc74258085)

[Fundamentos 3](#_Toc74258086)

[Programa cliente 4](#_Toc74258087)

[Programa servidor 5](#_Toc74258088)

[Ejecución 5](#_Toc74258089)

[Desarrollo 6](#_Toc74258090)

[Conclusión 8](#_Toc74258091)

[Código 1 Corriendo Server desde terminal 6](#_Toc74258107)

[Código 2 Ejecutando el programa desde la termina (CMD 6](#_Toc74258108)

[Código 3 Recibiendo información desde la termina (CMD) 7](#_Toc74258109)

[Código 4 Programa ejecutándose desde ambos servidores 7](#_Toc74258110)

# introducción

 sirven para interconectar dos sistemas a través de la red, sólo utilizando un número ip o nombre de host y un puerto determinado. La arquitectura utilizada en los sockets es la de Cliente/Servidor.

Con el uso de sockets en Java se pueden desarrollar muchos sistemas, como por ejemplos chats, videos juegos online y multijugador o incluso una simple página web [1].

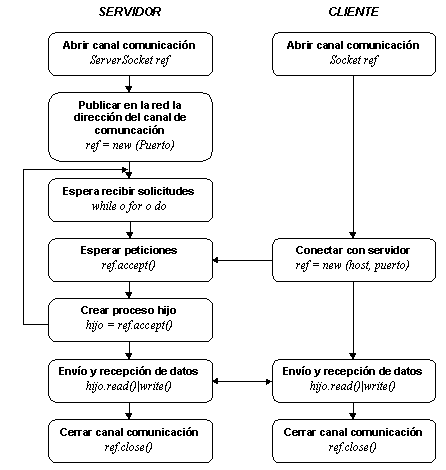
## Fundamentos

Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Más exactamente, un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información.

Fueron popularizados por Berckley Software Distribution, de la universidad norteamericana de Berkley. Los sockets han de ser capaces de utilizar el protocolo de streams TCP (Transfer Contro Protocol) y el de datagramas UDP (User Datagram Protocol).

Utilizan una serie de primitivas para establecer el punto de comunicación, para conectarse a una máquina remota en un determinado puerto que esté disponible, para escuchar en él, para leer o escribir y publicar información en él, y finalmente para desconectarse.

Con todas primitivas se puede crear un sistema de diálogo muy completo [1].



## Programa cliente

El programa cliente se conecta a un servidor indicando el nombre de la máquina y el número puerto (tipo de servicio que solicita) en el que el servidor está instalado [1].

Una vez conectado, lee una cadena del servidor y la escribe en la pantalla:

import java.io.\*;

import java.net.\*;

class Cliente {

static final String HOST = "localhost";

static final int PUERTO=5000;

public Cliente( ) {

try{

Socket skCliente = new Socket( HOST , Puerto );

InputStream aux = skCliente.getInputStream();

DataInputStream flujo = new DataInputStream( aux );

System.out.println( flujo.readUTF() );

skCliente.close();

} catch( Exception e ) {

System.out.println( e.getMessage() );

}

}

public static void main( String[] arg ) {

new Cliente();

}

}

## Programa servidor

El programa servidor se instala en un puerto determinado, a la espera de conexiones, a las que tratará mediante un segundo socket.

Cada vez que se presenta un cliente, le saluda con una frase "Hola cliente N".

Este servidor sólo atenderá hasta tres clientes, y después finalizará su ejecución, pero es habitual utilizar bucles infinitos ( while(true) ) en los servidores, para que atiendan llamadas continuamente [1].

Utiliza un objeto de la clase ServerSocket (skServidor), que sirve para esperar las conexiones en un puerto determinado (PUERTO), y un objeto de la clase Socket (skCliente) que sirve para gestionar una conexión con cada cliente.

Mediante un bucle for y la variable numCli se restringe el número de clientes a tres, con lo que cada vez que en el puerto de este servidor aparezca un cliente, se atiende y se incrementa el contador [1].

Para atender a los clientes se utiliza la primitiva accept() de la clase ServerSocket, que es una rutina que crea un nuevo Socket (skCliente) para atender a un cliente que se ha conectado a ese servidor.

Se asocia al socket creado (skCliente) un flujo (flujo) de salida DataOutputStream de escritura secuencial, en el que se escribe el mensaje a enviar al cliente [1].

El tratamiento de las excepciones es muy reducido en nuestro ejemplo, tan solo se captura e imprime el mensaje que incluye la excepción mediante getMessage() [1].

# Ejecución

Aunque la ejecución de los sockets está diseñada para trabajar con ordenadores en red, en sistemas operativos multitarea (por ejemplo Windows y UNIX) se puede probar el correcto funcionamiento de un programa de sockets en una misma máquina [1].

Para ellos se ha de colocar el servidor en una ventana, obteniendo lo siguiente:

>java Servidor

Escucho el puerto 5000

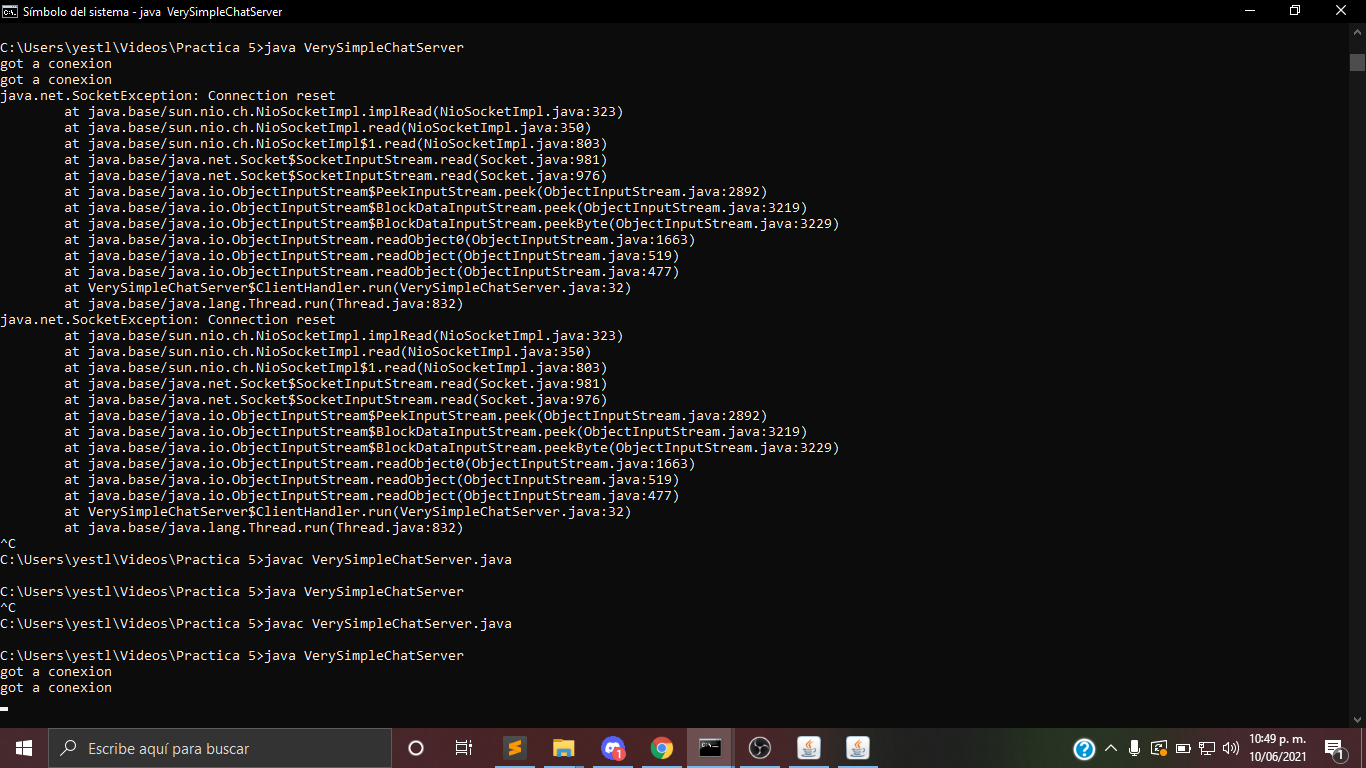
En otra ventana se lanza varias veces el programa cliente, obteniendo:

>java Cliente

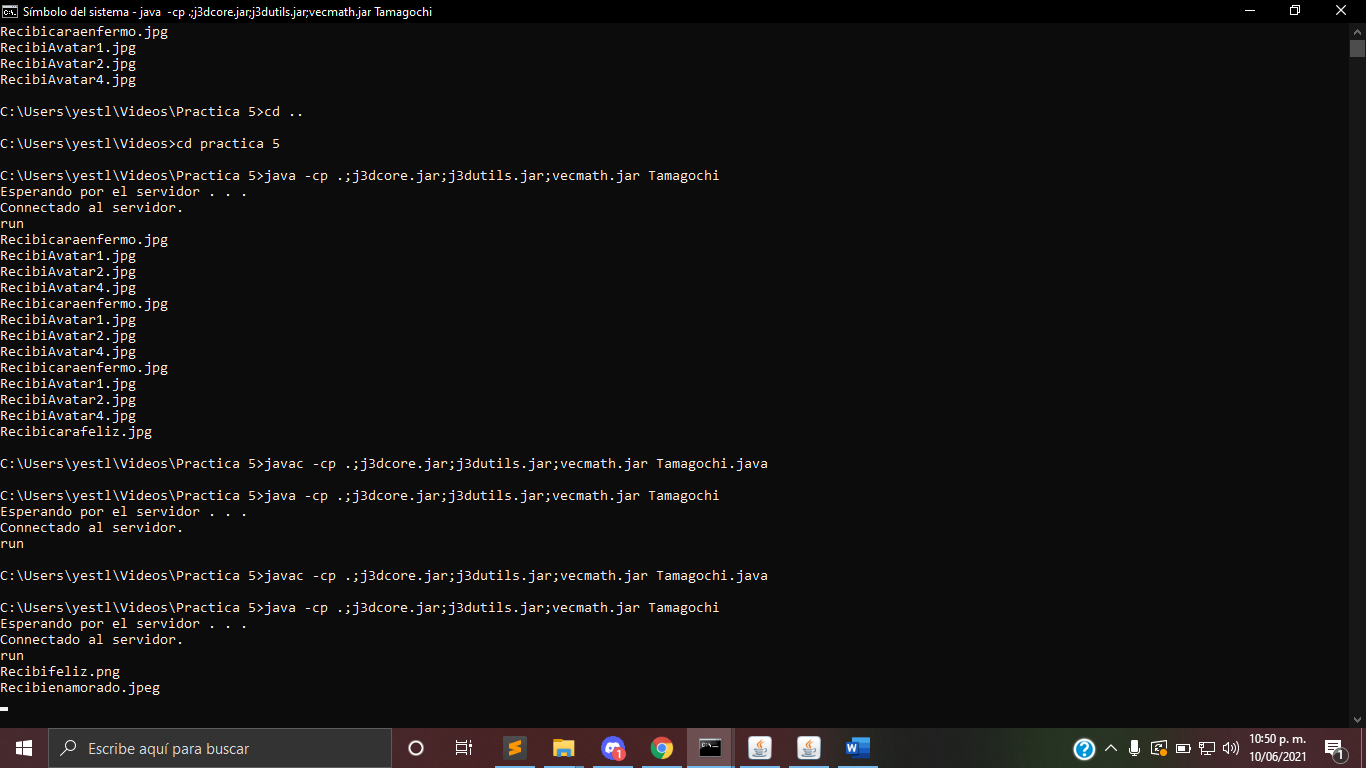
Hola cliente 1

>java cliente

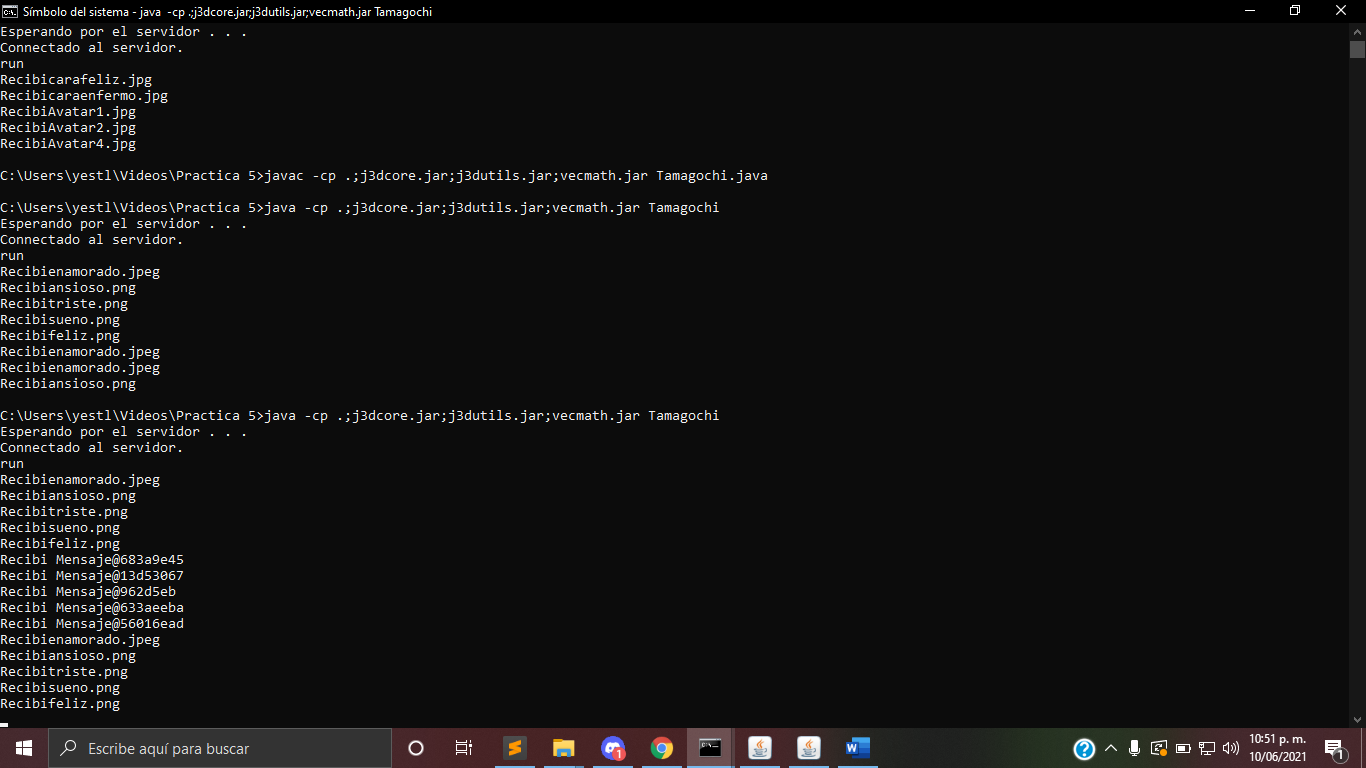
# Desarrollo



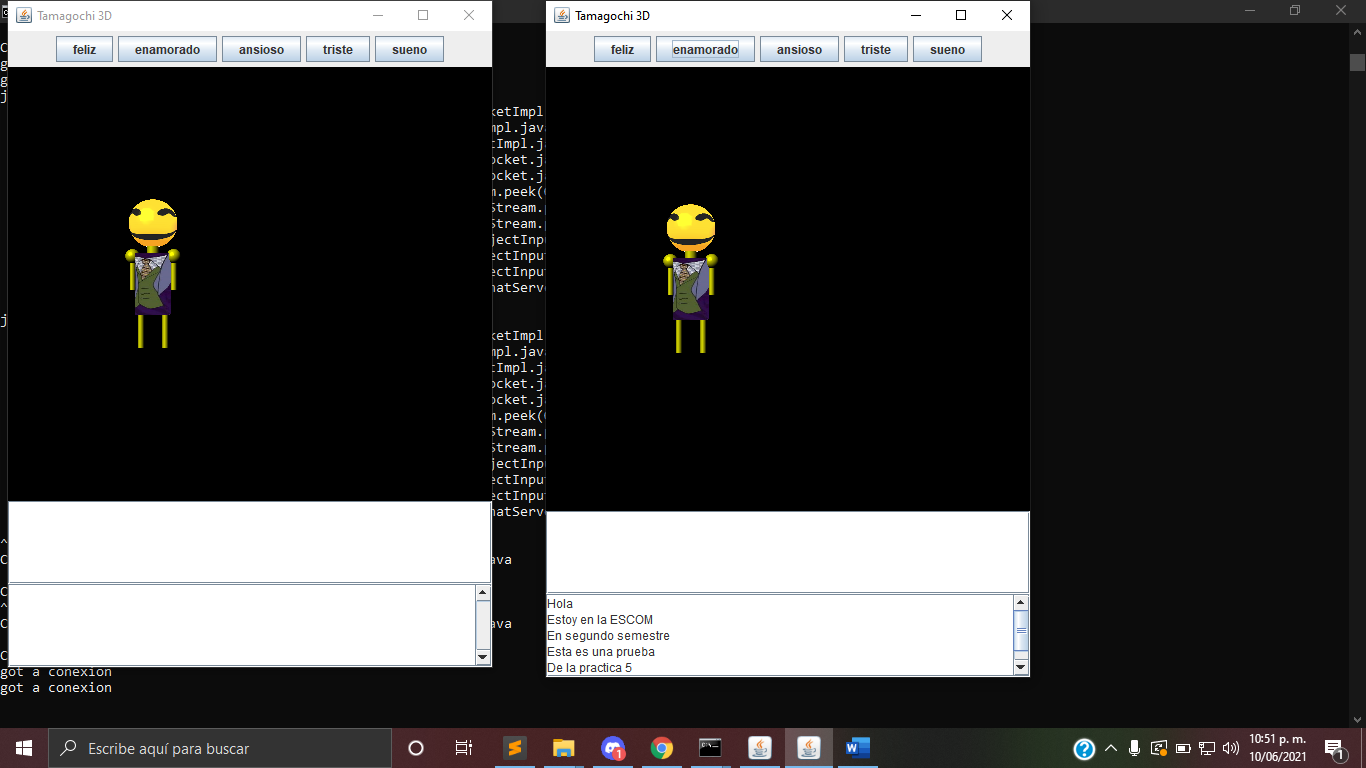
Código 1 Corriendo Server desde terminal



Código 2 Ejecutando el programa desde la termina (CMD



Código 3 Recibiendo información desde la termina (CMD)



Código 4 Programa ejecutándose desde ambos servidores

# Conclusión

Es importante saber que dentro de las distintas formas de escribir un Chatbot, lo que se buscaba en esta practica es tener presente y claro la forma de interactuar de cada uno de nuestros componentes.

La parte de la comunicación en red fue compleja ya que nunca había tenido la oportunidad de hacerlo, investigando un poco supongo que pude resolverlo de la mejor manera posible para mí, aunque no sea la más optima.

Fue nuevo y después de varios intentos, logre darme cuenta que para realizar los códigos, estos también se tenían que ejecutar en tres terminales diferentes, una donde se establece la conexión con el servidor y los otros dos para hacer dos tamagochis, donde si en uno se modifica el estado de ánimo, en el otro también se vera reflejado.