



INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)



PROGRAMACIÓN ORIENTADA A OBJETOS

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- SOCKETS CLIENTES

NÚMERO DE PRÁCTICA: 45

OPCIÓN: 1

- CHAT 3D

FECHA DE ENTREGA:

- 18/06/2021

GRUPO:

- 2CM11

Sockets Clientes

Introducción

Se desarrollo un programa haciendo uso de lo visto teóricamente en clase sobre los Sockets, consiste en el desarrollo con ayuda de Java 3D para hacer un personaje de tamagochi y que tenga diversos estados de ánimo, se hizo uso de muchas clases que permitieron hacer el cuerpo, el comportamiento, y el acceso a red.

Marco Teórico

Java fue pensado en ofrecer la máxima transparencia posible al trabajar en redes, a partir de abstraer los detalles específicos y diferentes de las diferentes redes involucradas en la comunicación.

Con objeto de liberar al programador Java de gran parte del trabajo de red de bajo nivel, Java proporciona dos formas básicas de trabajar en red: mediante Sockets y URL.

Los sockets son un mecanismo cliente/servidor que funciona a nivel de transporte (TCP Y UDP) y son los más utilizados. Consisten en comunicar un proceso entre dos puntos finales o puertos de dos máquinas conectadas mediante una red.

Los sockets representan una abstracción de conexión de red, que permite a dos programas, ejecutándose en dos máquinas o sistemas de comunicación diferentes, intercambiarse datos primitivos, en forma de streams o secuencias de bytes.

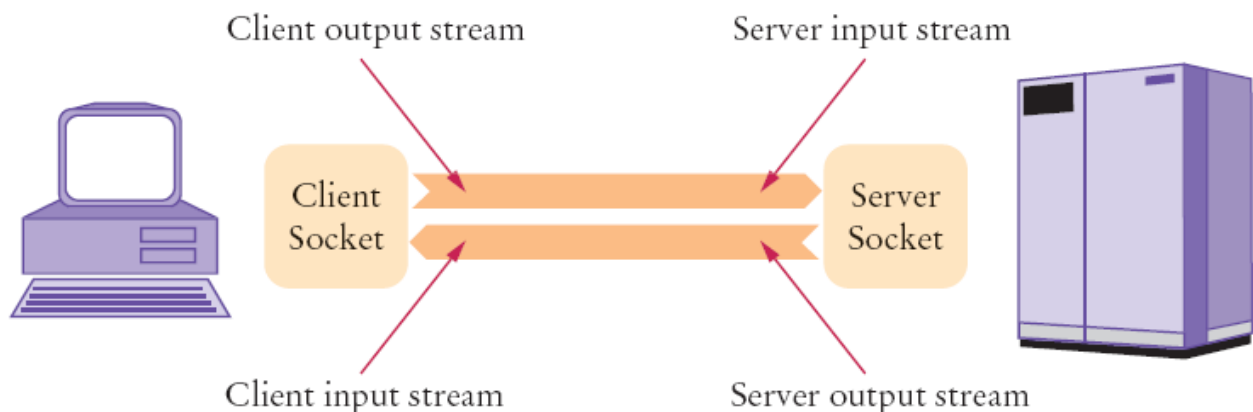
De esta forma, entre dos sockets, se abre un canal de comunicación, en la cual se escriben y leen datos en formas de streams.

OutputStream

Es una clase abstracta que representa el flujo de bytes de salida. Un output stream acepta bytes de salida que escribe en un dispositivo genérico. Un dispositivo de salida se considera apropiado mientras sea capaz de recibir o almacenar los datos que le llegan de un flujo de salida.

InputStream

Es una clase abstracta que representa un flujo de datos de entrada. Un input stream acepta bytes de entrada que lee de un dispositivo genérico. Un dispositivo de entrada se considera apropiado si puede trabajar como fuente de datos y coincide con los dispositivos de salida.



Desarrollo

- 1) Se hizo uso del paquete java.io es el encargado de gestionar las operaciones de entrada/salida. Entrada estándar sería System.in (es un objeto InputStream), salida estándar sería System.out y salida estándar de errores sería System.err (las salidas son objetos PrintStreams). Además, que a través de las clases del paquete java.net, los programas Java pueden utilizar TCP o UDP para comunicarse a través de Internet. Las clases URL, URLConnection, Socket, y SocketServer utilizan TCP para comunicarse a través de la Red.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.image.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.net.*;
import java.io.*;
```

- 2) Se hace uso de las clases VerySimpleCharServer y la clase anidada ClientHandler que implementa la interfaz Runnable donde se usa un ArrayList que va a contener flujos de salida para objetos (ObjectOutputStream). Además se declara un socket. Se va a implementar el método abstracto run (un hilo por cada cliente donde el hilo es creado por ClientHandler). En el constructor inicializamos la variable sock a partir del parámetro que recibe el constructor y luego se usa sock para obtener un flujo de entrada.

```

public class VerySimpleChatServer {
    ArrayList<ObjectOutputStream> clientObjectOutputStreams;// ArrayList que va a guardar ObjectOutputStream
    public class ClientHandler implements Runnable {    //clase ClientHandler que implementa la interfaz Runnable
        ObjectOutputStream writer;
        ObjectInputStream reader;
        Socket sock;//socket
        public ClientHandler(Socket clientSocket, ObjectOutputStream writer) {
            try {
                this.writer= writer;
                sock = clientSocket;
                reader = new ObjectInputStream(sock.getInputStream());
            } catch (Exception ex) {
                System.out.println("Exce Servidor reader " + ex);
                ex.printStackTrace();
            }
        }

        public void run() { //metodo Run
            Object obj;
            try {
                while (true) { //ciclo infinito
                    obj = (Object) reader.readObject();
                    tellEveryone(obj, writer);
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

    public static void main (String[] args) { //Metodo Main
        new VerySimpleChatServer().go();
    }

    public void go() {    //Metodo go
        clientObjectOutputStreams = new ArrayList<ObjectOutputStream>();
        try {

            try {
                ServerSocket serverSock = new ServerSocket(5000);
                while(true) {
                    Socket clientSocket = serverSock.accept();
                    ObjectOutputStream writer = new ObjectOutputStream(clientSocket.getOutputStream());
                    clientObjectOutputStreams.add(writer);
                    Thread t = new Thread(new ClientHandler(clientSocket, writer)); //Hilo
                    t.start(); // Metodo start
                    System.out.println("GOT A CONNECTION");
                }
            } catch (Exception ex) { // imprimir pila de llamadas
                ex.printStackTrace();
            }
        }

        public void tellEveryone(Object obj, ObjectOutputStream writerp) { //Metodo TellEveryone
            Iterator it = clientObjectOutputStreams.iterator();
            while(it.hasNext()) {
                try {
                    ObjectOutputStream writer = (ObjectOutputStream) it.next();
                    if(!writer.equals(writerp)){
                        writer.writeObject(obj);
                        writer.flush();
                    }
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        }
    }
}

```

- 3) Se crea el método run dentro del ClientHandler. Se hace uso de un ciclo infinito. La idea es que un objeto de tipo ClientHandler atiende a un cliente entonces en el método run vamos a estar esperando que el objeto llegue de la red. El método me va a devolver algo de tipo objeto, en la parte del reader.readObject() se va a tratar de leer un objeto de la red si esto tiene éxito entonces en obj guardamos el objeto que llegó de la red y luego el servidor manda a llamar a tellEveryone y le pasa el objeto que llegó de la red.

```

        public void run() { //metodo Run
            Object obj;
            try {
                while (true) { //ciclo infinito
                    obj = (Object) reader.readObject();
                    tellEveryone(obj, writer);
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}

```

- 4) Se crea el método main, donde crea un nuevo objeto de tipo VerySimpleChatServer y lo usamos para llamar al método go.

```

    public static void main (String[] args) { //Metodo Main
        new VerySimpleChatServer().go();
    }
}

```

- 5) Se crea el método go donde se crea un ServerSocket con el puerto 5000 (que está dentro de un try porque puede llamar a una excepción). El método go crea el objeto de tipo ArrayList donde se van a almacenar los flujos de salida. También se hace uso de un ciclo infinito. Hay que llamar a accept y bloquea el programa solo se desbloquea hasta que un cliente se conecte y entonces retorna un Socket que va a servir para comunicarse con el cliente.

```

    public void go() { //Metodo go
        clientObjectOutputStreams = new ArrayList<ObjectOutputStream>();
        try {
            ServerSocket serverSock = new ServerSocket(5000);
            while (true) {
                Socket clientSocket = serverSock.accept();
                ObjectOutputStream writer = new ObjectOutputStream(clientSocket.getOutputStream());
                clientObjectOutputStreams.add(writer);
                Thread t = new Thread(new ClientHandler(clientSocket, writer)); //Hilo
                t.start(); // Metodo start
                System.out.println("GOT A CONNECTION");
            }
        } catch (Exception ex) { // imprimir pila de llamadas
            ex.printStackTrace();
        }
    }
}

```

- 6) Se crea el método `tellEveryone`, este método recibe un objeto como parámetro. El `ArrayList` es un contenedor y entonces los contenedores se pueden recorrer. Obtenemos un iterador y este va a servir para recorrer el contenedor. El iterador tiene un método que devuelve cierto si es que aun quedan elementos que recorrer. La primera vez que llamemos al método `next` vamos a obtener el primer elemento que esté en el `ArrayList`. Cuando llamemos de nuevo a `next` obtenemos el segundo y así sucesivamente.

```
public void tellEveryone(Object obj, ObjectOutputStream writerp) { //Metodo TellEveryone
    Iterator it = clientObjectOutputStreams.iterator();
    while(it.hasNext()) {
        try {
            ObjectOutputStream writer = (ObjectOutputStream) it.next();
            if(!writer.equals(writerp)){
                writer.writeObject(obj);
                writer.flush();
            }
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

- 7) Se importan las librerías necesarias. Se ve que se está usando Java 3D.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.image.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.net.*;
import java.io.*;
```

- 8) Creamos la clase `Tamagochi` que extiende `JFrame` e implementa la interfaz `LeeRed`.

```
public class Tamagochi extends JFrame implements LeeRed {
```

- 9) En el constructor de la clase `Tamagochi` se inicializan las variables de instancia, los botones que en este caso se trata de 5 botones para cada estado de ánimo y se agregan con el método `add`. Se crea un objeto tipo `EventHandler` y se le coloca el nombre `eh`. Luego con el `addActionListener` en los botones vamos a suscribirnos a la acción más típica de estos y como parámetro le vamos a dar. Agregamos los botones a un panel, etc. Mandamos a llamar al método `setup3DGraphics`.

```

public Tamagochi(){
    super("Tamagochi 3D");
    turno=0;
    setSize(450, 500);
    GraphicsConfiguration config =
SimpleUniverse.getPreferredConfiguration();
    canvas3D = new Canvas3D(config);
    canvas3D.setSize(300, 400);
    eh = new EventHandler();
    bcambia=new JButton("Feliz");//boton feliz
    bcambia2=new JButton("Enfermo");//boton enfermo
    bcambia3=new JButton("Sonriendo");//boton sonriendo
    bcambia4=new JButton("Viajando"); //boton viajando
    bcambia5=new JButton("Confundido");//boton confundido
    bcambia.addActionListener(eh);
    bcambia2.addActionListener(eh);
    bcambia3.addActionListener(eh);
    bcambia4.addActionListener(eh);
    bcambia5.addActionListener(eh);
    jp1=new JPanel();
    jp1.add(bcambia);
    jp1.add(bcambia2);
    jp1.add(bcambia3);
    jp1.add(bcambia4);
    jp1.add(bcambia5);
    add("North", jp1);
    add("Center", canvas3D);
    setup3DGraphics();
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    setVisible(true);
    r=new Red(this);
    microChat = new MicroChat(r);
    add("South", microChat);
}

```

10) Se implementa el método main.

```

public static void main(String[] args) {
    new Tamagochi();
}

```

- 11) Se crea la clase Red donde se declara la variable cliente tipo Socket, un objeto de flujo de salida y un objeto de flujo de entrada. En el constructor se inicializa a la variable lr con el parámetro del constructor. Se llama al método setUpNetworking..

```
public class Red {
    private Socket cliente;
    private ObjectInputStream oisNet;
    private ObjectOutputStream oosNet;
    private int puerto=5000;
    private LeeRed lr;
    public Red(LeeRed lr) {
        this.lr=lr;
        setUpNetworking();
    }
    public void setUpNetworking() {
        int i=0;
        String host = JOptionPane.showInputDialog("Escriba dir.IP", "localhost");
        while(i==0){
            System.out.println("Esperando por el servidor . . ."); i=1;
            try {
                cliente=new Socket(host, puerto);
            } catch ( IOException e) {
                System.out.println("Fallo creacion Socket"); i=0;
            }
        }
        System.out.println("Conectado al servidor.");
        try {
            oisNet = getOISNet(cliente.getInputStream());
            oosNet = getOOSNet(cliente.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Error al crear los fujos de objeto"+e);
        }
        (new Thread( new IncomingReader(lr, oisNet) )).start();
    }
    public void escribeRed(Object obj) {
        try {
            oosNet.writeObject(obj);
            oosNet.flush();
        }
    }
}
```

<


```

    }
    ObjectOutputStream getOOSNet(OutputStream os) throws IOException {
        return new ObjectOutputStream(os);
    }
    ObjectInputStream getOISNet(InputStream is) throws IOException {
        return new ObjectInputStream(is);
    }
}

```

- 12) Se declara el método `escribeRed`, este recibe un objeto como parámetro. Usamos el método `WriteObject` de la clase `oosNet`, le pasamos el objeto y luego se llama al método `flush` donde se vacía el buffer.

```

    public void escribeRed(Object obj) {
        try {
            oosNet.writeObject(obj);
            oosNet.flush();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    ObjectOutputStream getOOSNet(OutputStream os) throws IOException {
        return new ObjectOutputStream(os);
    }
    ObjectInputStream getOISNet(InputStream is) throws IOException {
        return new ObjectInputStream(is);
    }
}

```

- 13) Creamos la clase `IncomingReader` que implementa interfaz `Runnable`, donde se declara una variable `lr` tipo `LeerRed` y un objeto de flujo de salida y en el constructor se le da como parámetro `LeerRed lr` y `ObjectInputStream oisNet`, Se inicializan las variables `lr` y `oisNet`.

```

    public class InComingReader implements Runnable {
        private LeerRed lr;
        private ObjectInputStream oisNet;
        public InComingReader(LeerRed lr, ObjectInputStream oisNet){
            this.lr=lr;
            this.oisNet=oisNet;
        }
        ...
    }

```

- 14) Se implementa el método run donde se va a usar un ciclo infinito. Donde lr es un objeto de una clase que implementa la interfaz LeeRed.

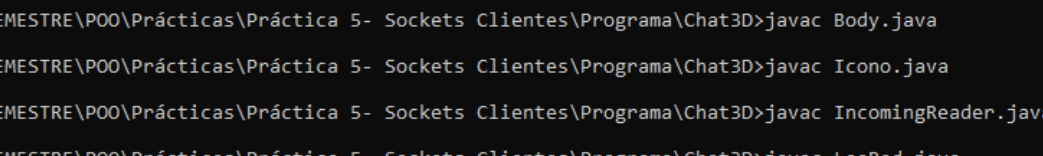
```
public void run(){
    Object obj=null;
    int j,k=0;
    System.out.println("run");
    for(;;){
        j=0;
        try {
            obj=oisNet.readObject();
        } catch (IOException e) {
            System.out.println("IO ex"+e);
        }
        j=1;
    } catch (ClassNotFoundException ex) {
        System.out.println("Class no found"+ex);
        j=1;
    }
    if (j==0) {
        lr.leeRed(obj);
    }
}
}
}
```

- 15) Se crea la clase Body donde se va a hacer la parte del diseño del cuerpo del muñeco, esto de diseño de objetos en Java3D.

```
public class Body{
    private BranchGroup group;
    private static Texture texture;
    private Appearance appearance;
    private TextureLoader loader;
    TransformGroup rotacion= new TransformGroup();
    TransformGroup tran;
    Appearance ap;
    Frame frame1;
    Sphere cab;


    public Sphere getCab(){
        return cab;
    }
    public TransformGroup getTransformGroup(){
        return tran;
    }
    public Body(float x, float y, float z,String nombre_mostrar,boolean movimiento,Frame frame,String img){
```

16) Procedemos a compilar cada una de las clases



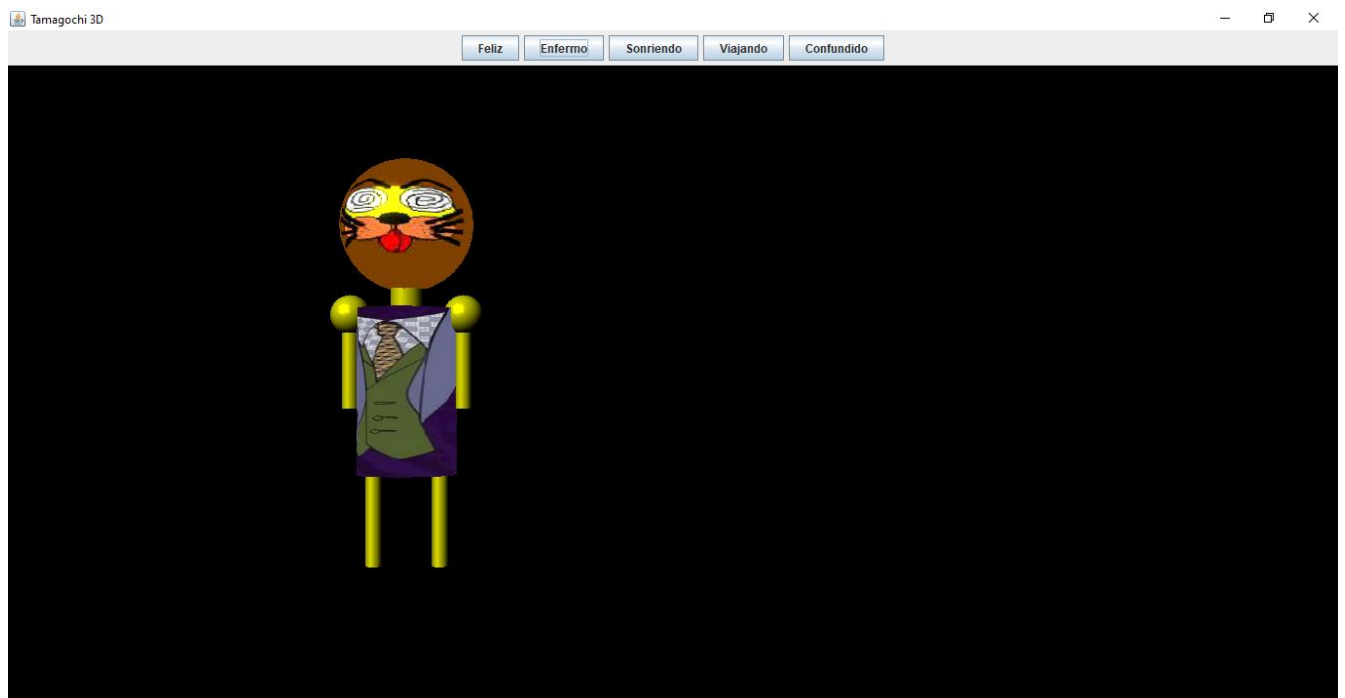
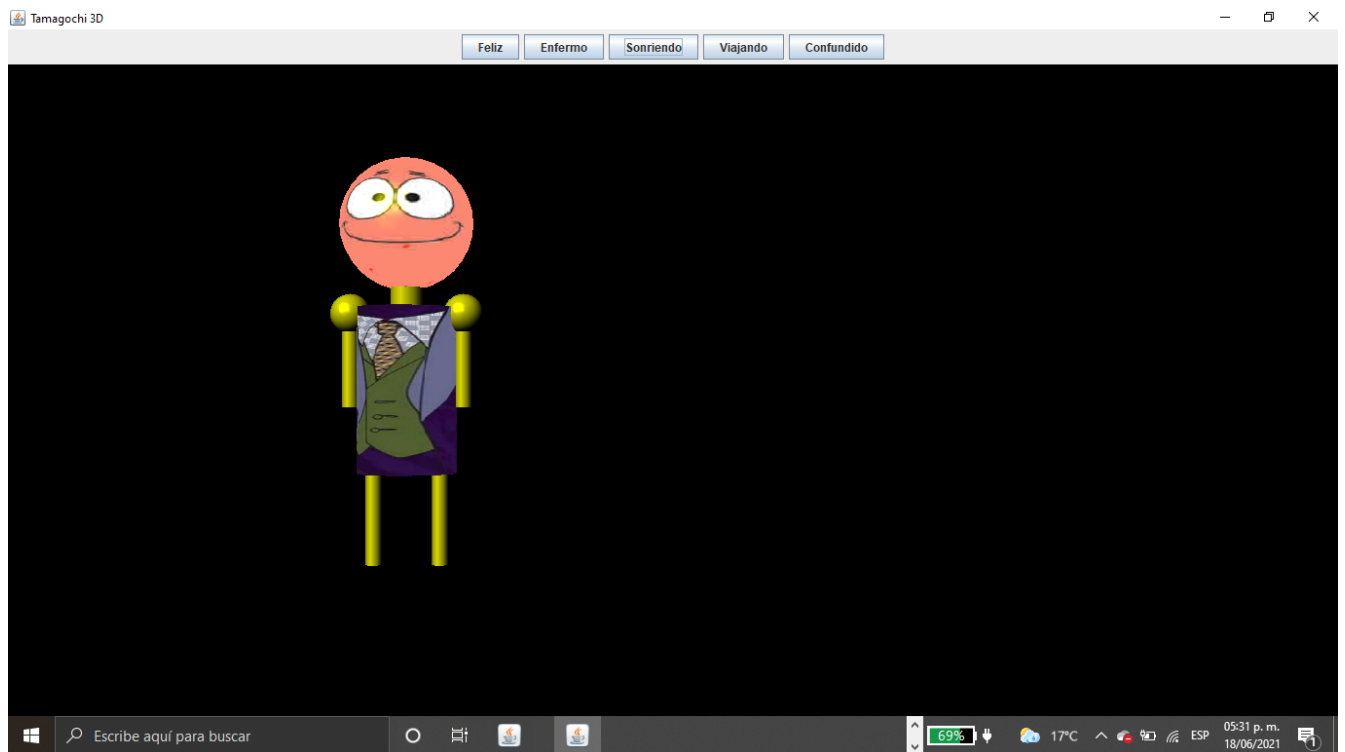
```
Símbolo del sistema - java Tamagochi

D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac Body.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac Icono.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac IncomingReader.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac LeeRed.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac Mensaje.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac MicroChat.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac Red.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac Tamagochi.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>javac VerySimpleChatServer.java
D:\CUARTO SEMESTRE\P00\Prácticas\Práctica 5- Sockets Clientes\Programa\Chat3D>java Tamagochi
```



Le damos a aceptar y procedemos a ejecutar el servidor

[illegible]



Conclusión

En conclusión, se logró desarrollar la conexión del cliente al colocar la dirección IP que se solicita se logra tener en la consola el mensaje que hay una conexión.