INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

ALUMNO: Meza Vargas Brandon David.

PRÁCTICA: Práctica No. 5.

TEMA: Sockets clientes.

OPCIÓN: Opción 1, Chat3D

FECHA: 14-dic-2020

GRUPO: 2CM1

MATERIA: Programación Orientada a Objetos

INTRODUCCIÓN

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.

DESARROLLO

La práctica consistió en añadir 3 estados de ánimo mas al tamagochi, de tal forma que ahora tendrá 5, de los cuales cada uno tiene su botón para ir cambiándolo.

Para el correcto funcionamiento de nuestro programa se hizo uso de varias clases, así como de una interfaz, en primero lugar tenemos la clase VerySimpleCharServer, la cual actúa como servidor de nuestro programa, este servidor tiene la capacidad de recibir un objeto de un cliente y enviarlo a los demás clientes.

Dentro de esta clase tenemos una primera clase llamada ClientHandler que implementa la interfaz Runnable, aquí se inicializan los flujos de entrada y salida, además del socket del cliente, el método run lee el objeto que manda el cliente, ver figura 1.

```
public class ClientHandler implements Runnable {
    //PrintWriter writer;
    ObjectOutputStream writer;
    //BufferedReader reader;Pa
    ObjectInputStream reader;
    Socket sock;

    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();
       }
     } // close constructor

   public void run() {
      Object obj;

      try {
        while (true) {
          obj = (Object) reader.readObject();
          //System.out.println("read " + pub);
          tellEveryone(obj, writer);
        }
      } catch(Exception ex) {ex.printStackTrace();}
  } // close run
// close inner class
```

Figura 1. Clase ClientHandler

El método go asigna el puerto del servidor, además de inicializarlo, gracias a un hilo, permite que varios clientes se puedan conectar, ver figura 2.

```
public void go() {
  //clientOutputStreams = new ArrayList<PrintWriter>();
  clientObjectOutputStreams = new ArrayList<ObjectOutputStream>();
  try {
  ServerSocket serverSock = new ServerSocket(5000);

  while(true) {
    Socket clientSocket = serverSock.accept();
    //PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
    //clientOutputStreams.add(writer);
    ObjectOutputStream writer = new ObjectOutputStream(clientSocket.getOutputStream());
    clientObjectOutputStreams.add(writer);
  Thread t = new Thread(new ClientHandler(clientSocket, writer));
  t.start();
  System.out.println("Conexion Establecida");
}
  // now if I get here I have a connection
 }catch(Exception ex) {
    ex.printStackTrace();
 }
```

Figura 2. Método go()

El método llamado tellEveryone() se encarga de enviar el objeto recibido por un cliente a los demás clientes, ver figura 3.

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

    } // end while
} // close tellEveryone
```

Figura 3. Método tellEeveryone().

La clase Red se encarga de realizar la conexión con el servidor, conteniendo sus respectivos try and catch, ver figura 4.

```
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 direccion IP del server", "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("Connectado 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();
    } 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);
}
```

Figura 4. Clase Red.

La clase IncomingReader, se encarga de leer un flujo de entrada, generalmente hablando un objeto de tipo Chat3D, siendo más específico un objeto de una clase que implementa la interfaz LeeRed, ver figura 5.

```
public class IncomingReader implements Runnable {
private LeeRed lr;
private ObjectInputStream oisNet;
public IncomingReader(LeeRed lr, ObjectInputStream oisNet){
        this.lr=lr;
        this.oisNet=oisNet;
}
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);
                }//if
        }//for
}//run
}
```

Figura 5. Clase LeeRed.

En nuestro programa tenemos una clase llamada Body, la cual se encarga de crear a nuestro personaje y de acuerdo a los parámetros que reciba de nuestra clase char3D, será como irán cambiando los estados de ánimo del tamagochi, aquí usamos elementos vistos en la practica 3 de java3D.

El método mas importante en esta clase es el llamado changeTextureCab, pues este será el encargado de cambiar el estado de animo de nuestro tamagochi, ver figura 6.

```
public void changeTextureCab(Texture texture, String image) {
    loader = new TextureLoader(image, "RGB",frame1);
    texture = loader.getTexture();
    texture.setBoundaryModeS(Texture.CLAMP_TO_BOUNDARY);
    texture.setBoundaryModeT(Texture.CLAMP_TO_BOUNDARY);
    texture.setBoundaryColor(new Color4f(0.0f, 1.0f, 0.5f, 0f))
    TextureAttributes texAttr = new TextureAttributes();
    texAttr.setTextureMode(TextureAttributes.REPLACE);
    ap.setTextureAttributes(texAttr);
    ap.setTexture(texture);
}
```

Figura 6. Método changeTextureCab() de nuestra clase Body.

Nuestra clase principal es Chat3D que extiende a Frame e implementa la interfaz LeeRed, primeramente, en la figura 7 vemos las variables que se usarán:

```
public class Chat3D extends Frame implements LeeRed{
    Canvas3D canvas3D;
    Appearance ap;
    Texture feliz;
    Texture enfermo;
    int cont;
    private static Texture texture;
    private Appearance appearance;
    private String estados[]={"C:\\Users\\PC\\Documents\\NetBeansProjects\\Tamagochi\\src\\images\\carafeliz.jpg","C:\\Users\\PC\\Documents\\NetBeansProjects\\Tamagochi\\src\\images\\caraenfermo
                    "C:\\Users\\PC\\Documents\\NetBeansProjects\\Tamagochi\\src\\images\\triste.jpg","C:\\Users\\PC\\Documents\\NetBeansProjects\\Tamagochi\\src\\images\\enamorado.jpg",
                    "C:\\Users\\PC\\Documents\\NetBeansProjects\\Tamagochi\\src\\images\\enojado.jpg"};
    private TextureLoader loader;
    Body b;
    private Red r;
    Panel p;
    Button bfeliz, benfer, bena, beno, btris;
    EventHandler eh;
```

Figura 7. Variables clase Chat3D.

En el constructor inicializamos los elementos de nuestro Frame, ver figura 8.

```
public Chat3D() {
    super("Practica 5");
    cont =0;
    setResizable(false); setSize(500, 400);
    GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
    canvas3D = new Canvas3D(config);
    eh = new EventHandler();

    bfeliz=new Button("Feliz");
    bfeliz.addActionListener(eh);
    benfer=new Button("Enfermo");
    benfer.addActionListener(eh);
    bena=new Button("Enamorado");
    bena.addActionListener(eh);
    beno=new Button("Enojado");
    beno.addActionListener(eh);
    btris=new Button("Triste");
    btris.addActionListener(eh);
    p=new Panel();
    p.add(bfeliz); p.add(benfer); p.add(beno); p.add(bena); p.add(btris);
    add("North", p); add("Center",canvas3D);
    addWindowListener(eh);
    setup3DGraphics();
    setVisible(true);
    r=new Red(this);
}
```

Figura 8. Elementos Frame.

En la clase EventHandler tenemos el método actionPerformed, el cual se encargará de recibir el botón que fue pulsado y gracias a ifs, determinar cual fue el estado de animo seleccionado y así, cambiarlo, ver figura 9.

```
class EventHandler extends WindowAdapter implements ActionListener {
  public void actionPerformed(ActionEvent e) {
     Button btn=(Button)e.getSource();
   if(btn==bfeliz){ cont=0; }
                  else if (btn==benfer){cont=1;}
                  else if (btn==btris){cont=2;}
                  else if (btn==bena){cont=3;}
                  else if (btn==beno){cont=4;}
     b.changeTextureCab(texture, estados[cont]);
     r.escribeRed(new Icono("Tamagochi", cont));
  }
```

Figura 9. clase Event Handler.

Para ejecutar nuestro programa, debemos de compilar nuestro servidor y después ejecutarlo, además de compilar las clases que estén presentes, para finalmente correr nuestra clase Chat3D que es aquella que contiene el main.

La salida del código lo podemos ver en la figura 10.

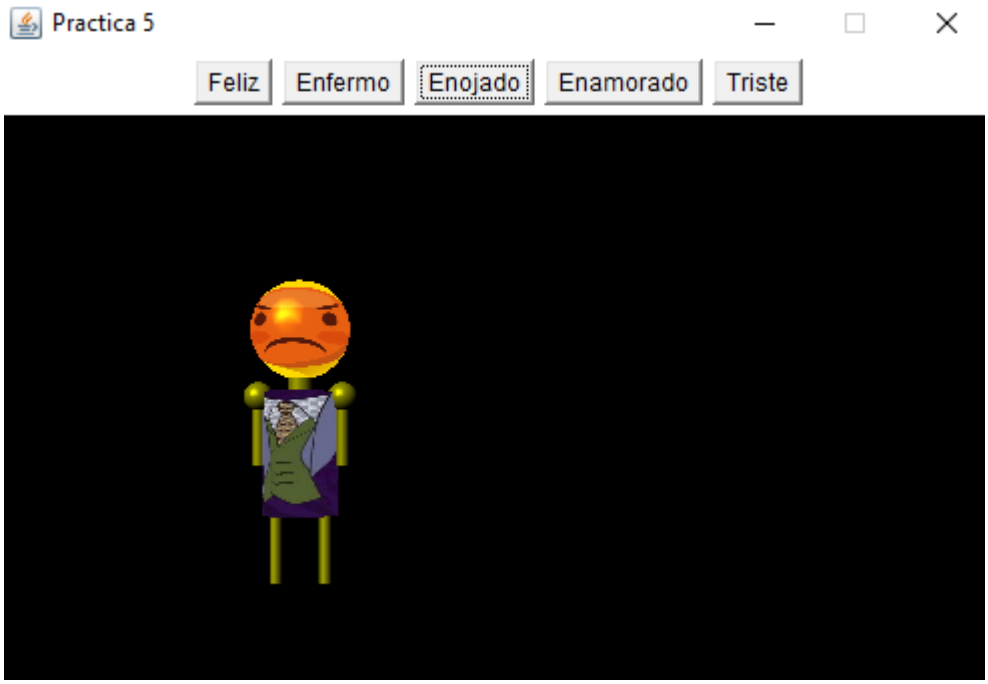

Figura 10. Salida de nuestro programa.

Nuestro tamagochi con otro estado de animo lo podemos ver en la figura 11:

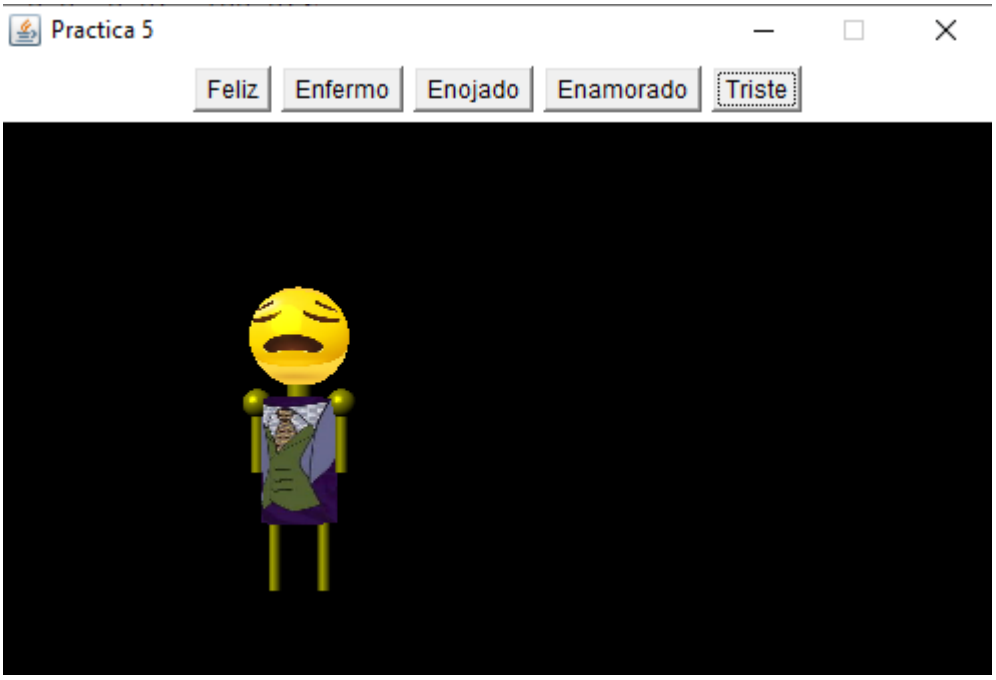

Figura 11. Tamagochi con otro estado de ánimo.

CONCLUSIÓN

Como vimos en la introducción y desarrollo de esta practica los sockets nos ayudan para comunicar un cliente con un servidor.

Puedo concluir que tanto el cliente como el servidor pueden leer o escribir en el socket, y estos mecanismos de comunicación pueden ser refinados cambiando la implementación de los sockets. Además, que el protocolo a utilizar va a depender de la aplicación cliente/servidor que se esté escribiendo.

Una practica interesante donde me adentre a la programación con sockets en java.