Instituto Politécnico Nacional

Escuela Superior de Cómputo

Programación Orientada a Objetos

Reporte de Práctica #5

Sockets Clientes

Profesor: Roberto Tecla Parra

Alumno: Calva Hernández José Manuel

Grupo: 2CM3

Moviendo objetos de una maquina a otra usando un servidor como intermediario.

Para esa practica dispone del servidor VerySimpleChatServer el cual tiene la capacidad de recibir un objeto de un cliente y enviarlo a los demás clientes.

ChatBot (Carpeta estatiProgBarRedSimBot)

Modificar el programa que cambia el estado de animo del tamagochi en la maquina local y la maquina remota de modo que actué como un ChatBot con al menos 10 preguntas y respuestas predefinidas. En este caso se debe enviar un objeto que contenga una cadena (mensaje) al servidor. En el problema 1 el objeto que se mueve entre maquinas es de la clase Icono dicha clase contiene un entero que es el dato que nos interesa enviar de una maquina a otra.

Ejemplos de preguntas tipo y respuestas tipo

En que ciudad vives?

D.F.

Cuantos años tienes?

20

En que escuela estudias?

ESCOM

Objetivos

- ➢ Entender la funcionalidad del servidor como intermediario para el recibo y envío de objetos.
- ➢ Entender cada uno de los pasos del protocolo para socket cliente, con el objetivo de realizar una correcta conexión entre los clientes y el servidor.
- ➢ Mover objetos de una máquina a otra usando un servidor como intermediario.
- ➢ Los objetos que se enviarán a través del servidor, los cuales son recibidos por un cliente, se deberán mostrar en el/los demás clientes.

Desarrollo

Se importaron las librerías necesarias, incluyendo las aplicables para Java 3D, con el objetivo de utilizar los métodos necesarios para la práctica. Se creó la clase Tamagochi que extiende a JFrame e implementa la interfaz LeeRed. Se crearon las variables necesarias para el corrimiento de Tamagochi, la cual actuaría como cliente, y algunas de las más destacadas fueron el arreglo de preguntas[] y respuestas[], el cual contendría los flujos de conversación entre ambos bots, red, el sería vital para la actividad entre el servidor y los clientes, los JPanel que contendrían a los objetos (Botones el primero y el segundo al Avatar del Tamagochi),y los JButton que contendrían la acción para cambiar a cada una de las imágenes en el Avatar.

```
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.*;
```

```
public class Tamagochi extends JFrame  implements LeeRed {
  static String preguntas []= { "COMO TE LLAMAS","DONDE VIVES","CUANTOS AÑOS TIENES","DONDE ESTUDIAS",
  static String respuestas []= {"MI NOMBRE ES TAMAGOCHI","EN LA COMPU","1045","INTERNET","LABORATORIO"
  private Canvas3D canvas3D;
  private Appearance ap;
  private static Texture texture;
  private JPanel jp1, jp2;
  private JTextField leer;
  private JLabel responder;
  private JButton bcambia;
  private EventHandler eh;
```

Posteriormente se creó el método constructor Tamagochi (para el cliente) el cual superponía a Tamagochi 3D, se estableció el tamaño de ancho y alto, y para la configuración de los gráficos se crearon los nuevos objetos de tipo JButton que serían utilizados para los cambios de estado, a su vez se les aplicó el ActionListener, a su vez fueron agregados uno por uno al Panel 1, se puso en la zona Norte de a pantalla al Panel 1 y en el Centro al Canvas, el cual contendría al Avatar, por último se creó un nuevo Microchat, el cual sería puesto a Sur del Frame.

```
public Tamagochi(){
 super("Tamagochi 3D");
 turno=0;
 //setResizable(false);
 setSize(400, 500);
 GraphicsConfiguration config =
 SimpleUniverse.getPreferredConfiguration();
 canvas3D = new Canvas3D(config);
 canvas3D.setSize(300, 400);
 eh = new EventHandler();
 leer = new JTextField("");
 responder = new JLabel("    ");
 bcambia=new JButton("Cambiar");
 bcambia.addActionListener(eh);

 jp1=new JPanel();
 jp1.setLayout(new GridLayout(3,1,5,5));
 jp1.add(leer);
 jp1.add(responder);
 jp1.add(bcambia);
 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);
}
```

Posteriormente se crearía un método llamado setup3DGraphics, en el cual se establecerían los parámetros adecuados para poner al Avatar del Tamagochi dentro del canvas 3D, y a su vez este empieza con una imagen llamada carafeliz, todo esto se pondría en el Branch Group y el Avatar sería seteado con vectores de posición.

```
void setup3DGraphics(){
    BranchGroup group=null;
   DirectionalLight light1;
   SimpleUniverse universe = new SimpleUniverse(canvas3D);
   universe.getViewingPlatform().setNominalViewingTransform();
   if(avatar== 0){
     movi=new Esfera(this);
     movi.changeTextureCab(texture, "Arizona.jpg");
     group = movi.myBody();
   }
   if(avatar== 1){
     movi=new Esfera(this);
     movi.changeTextureCab(texture, "carafeliz.jpg");
     group = movi.myBody();
   }
   if(avatar== 2){
     movi=new Body(-0.4f,0.0f,0.0f,"",true, this, "Avatar1");
     movi.changeTextureCab(texture, "carafeliz.jpg");
     group = movi.myBody();
   }
   if(avatar== 3){
     movi=new BodyBob(-0.4f,0.0f,0.0f,"",true, this, "bob-esponja.jpg");
     group = movi.myBody();
   }
   if(avatar== 4){
     movi=new BodyZoey(-0.4f,0.0f,0.0f,"",true, this, "cabeza.png");
     group = movi.myBody();
   }
   if(avatar== 5){
     movi=new Stan(-0.4f,0.0f,0.0f,"",true, this, "Avatar1", null);
     group= movi.myBody();
   }
   BoundingSphere bounds =new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);
   Background fondo=new Background();
   fondo.setColor(1.0f,1.0f,1.0f);
   fondo.setApplicationBounds(bounds);
   group.addChild(fondo);
//SE DA LUZ A TODO EL ESCENARIO
   AmbientLight luz= new AmbientLight();
   luz.setInfluencingBounds(bounds);
   group.addChild(luz);
   universe.addBranchGraph(group);
}
```

Luego se crearía el método main, el cual crearía un nuevo Tamagochi, contendría a la clase EventHandler, y esta su vez implementaría ActionListener, se crearía posteriormente el ActionPerformed en donde se programarían los eventos de los botones, esto de acuerdo al turno que se especificaría con un cliqueo en cada uno de ellos, y de acuerdo al nombre del botón se le asignaría cierto valor al turno para así poder setear las imágenes correspondientes del nombre del botón al Avatar (cambiando de textura).

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


class EventHandler implements ActionListener {
  public void actionPerformed(ActionEvent e1) {
      Object obj=e1.getSource();
      if(obj instanceof JButton){
       JButton btn=(JButton)e1.getSource();
       if(btn==bcambia){
        String s = leer.getText();
       r.escribeRed(new Mensaje("Tamagochi",s));}
      }
  }
}
```

Por último, se creó el método leerRed en el cual se mostraría en consola, por parte del servidor, el mensaje Recibí y el nombre de la imagen, para el caso de haber escrito un mensaje en el Microchat el mensaje sería similar solo que con el nombre del mensaje. También se creó el método findMatch para preguntas y respuestas.

```
public void leeRed(Object obj){
    if(obj instanceof Mensaje){
      System.out.println("Recibi "+(Mensaje)obj);
      microChat.recibir((Mensaje)obj);
    }
  }
```

En la clase MicroChat se modificó el flujo de datos para poder intercambiar mensajes entre dos bots.

```
public class MicroChat extends JPanel implements ActionListener{
private JTextArea incoming;
private JTextField outgoing;
private Red r;
public MicroChat (Red r) {
    super();
        this.r=r;
        setLayout(new GridLayout(2,1));
        incoming = new JTextArea(5,20);  // was 15,50
        incoming.setLineWrap(true);
        incoming.setWrapStyleWord(true);
        incoming.setEditable(false);
        JScrollPane qScroller = new JScrollPane(incoming);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        outgoing = new JTextField(15);
        outgoing.addActionListener(this);
    add(outgoing);
    add(qScroller);
}
public void actionPerformed(ActionEvent e) {
        System.out.println("actionPerformed if ("+outgoing.getText()+")");
    r.escribeRed(new Mensaje("Escom","jojo", outgoing.getText()));
    outgoing.setText("");
}
public void recibir(Mensaje mensa){

   coming.append(findMatch(mensa.getTexto()) + "\n");
}

 static String findMatch(String str) {
        String result = "";
        for(int i = 0; i < preguntas.length; ++i) {
            if(str.toLowerCase()==preguntas[i].toLowerCase()) {
                result = respuestas[i];
                break;
            }
        }
        return result;
    }

}
```

En la parte del mensaje, se modificó el constructor para evitar uno de los parámetros de la siguiente forma:

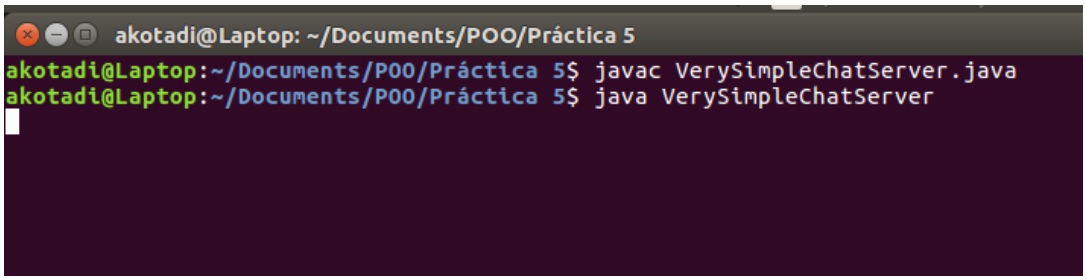
```
public Mensaje(String programa, String texto){
    this.programa=programa;
    this.texto=texto;
}
public String getPrograma(){
            return programa;
}
public String getNick(){
            return nick;
}
public String getTexto(){
        return texto;
}
```

Para la parte del servidor se usó una clase llamada VerySimpleChatServer la cual efectuaría cada uno de los pasos para el Socket Servidor, como lo vimos en clase.

Así como la clase Red fue modificada de la siguiente forma:

```
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");
   String host = "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);
}
}
```

Para probar el código debemos ejecutar tres terminales, una de ellas simulará el servidor mientras las otras dos se comunican por medio de él como se muestra a continuación:



El servidor se queda activo (en espera) para poder procesar los flujos de datos.

Las dos terminales mantienen comunicación por medio del servidor.

Conclusiones

Para lograr una buena comunicación entre dos o más clientes y se muestren las actividades de entrada y salida, es necesario que haya un intermediario llamado Servidor, el cual podrá manipular el paso de objetos e información de tal manera que todos muestren las mismas características en su frame.