PROGRAMACIÓN DE SERVICIOS Y PROCESOS

UT03. Programación de Comunicaciones en Red

Práctica a entregar

Práctica 1. Creación de un aplicación de Chat con TCP.

Una situación típica de un servidor que atiende a múltiples clientes es un servidor de chat. Vamos a construir un chat sencillo que pueda atender a varios clientes a la vez, cada cliente será atendido en un hilo de ejecución; en ese hilo se recibirán sus mensajes y se enviarán al resto. La idea básica en el servidor es la siguietne:

- Al iniciar el servidor se muestra una pantalla donde se visualiza el número de clientes que actualmente están conectados al chat y la conversación mantenida entre ellos. La conversación de chat se va visualizando enun TextArea. El botón Salir finaliza el servidor de chat, Figura 1.
- El servidor se mantiene a la escucha (en un puerto pactado) de cualquier petición de un cliente para conectarse.
- El servidor acepta al cliente, guarda en un array de sockets el que se acaba de crear. Este array se usará en el hilo de ejecución para enviar la conversaión del chat a todos los clientes conectados.
- Cuando se conecta un cliente se incrementa en un contados el número de conexiones actuales, que viene dado por la variable ACTUALES, si se desconecta un cliente se decreamenta. Otro contador, que viene dado por la varialbe CONEXIONES, se usará para contar las conexiones de clientes, el máximo de conexiones viene dado por la variable MAXIMO.
- Se lanza un hilo de comunicación con el cliente (programa HiloServidor). Por el hilo se reciben y envían los mensajes de los clientes. Si el cliente cierra la comunicación, el hilo se rompe y se corta la comunicación con ese cliente.
- Se adminte hasta un máximo de 10 conexiones.

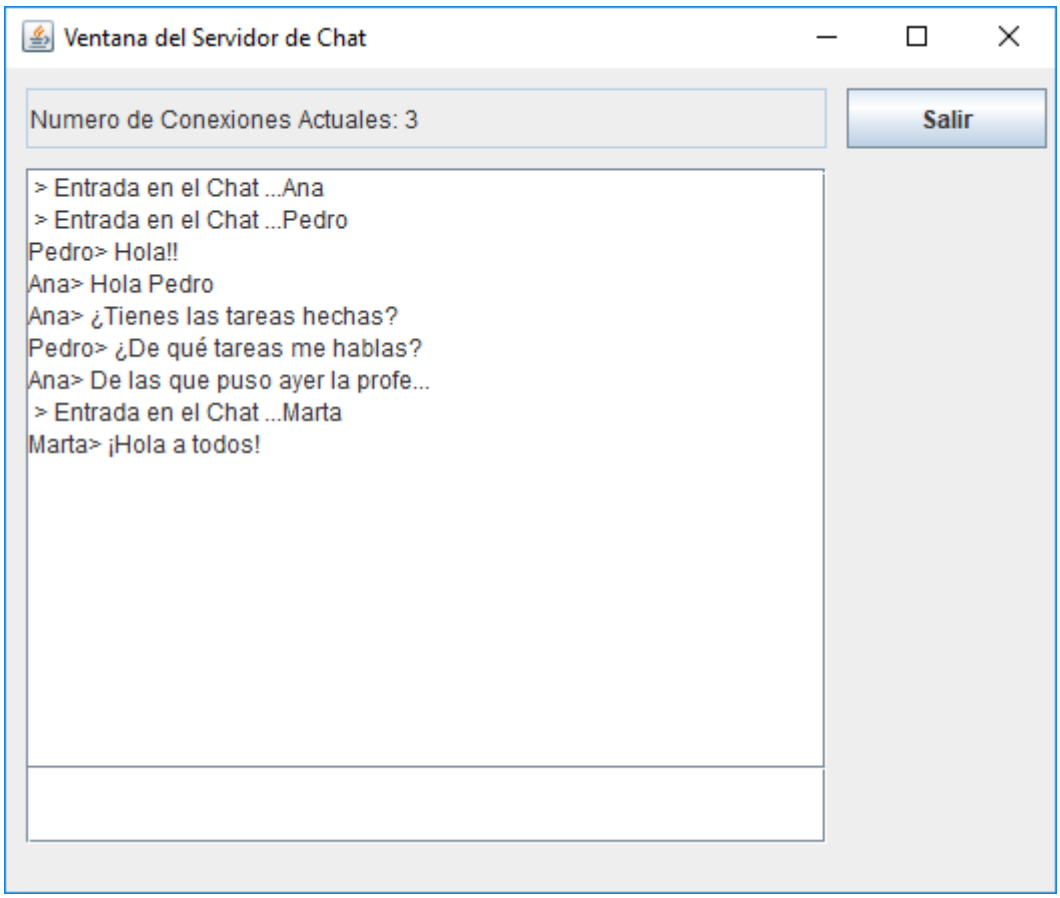

***Figura 1**. Aplicación Servidor de Chat*

El programa servidor, **ServidorChat**, escuchará en el puerto *5050*. En primer lugar se definen las variables y campos de la pantalla:

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ServidorChat extends JFrame implements ActionListener {

    // Atributos
    private static final long serialVersionUID = 1L;
    static ServerSocket servidor;
    static final int PUERTO = 5050; // Puerto por el que escucha el servidor
    static int CONEXIONES = 0;      // Cuenta el numero de conexiones
    static int ACTUALES = 0;        // Numero de conexiones actuales activas
    static int MAXIMO = 10;         // Maximo de conexiones permitidas

    static JTextField mensaje = new JTextField("");
    static JTextField mensaje2 = new JTextField("");
    private JScrollPane scrollpanel;
    static JTextArea textarea;
    JButton salir = new JButton("Salir");
```

```
    static Socket tabla[] = new Socket[MAXIMO]; // Almacena sockets de clientes
```

Desde el constructor se prepara la pantalla:

```
    // Constructor
    public ServidorChat() {

        super("Ventana del Servidor de Chat");
        setLayout(null);
        mensaje.setBounds(10, 10, 400, 30);
        add(mensaje);
        mensaje.setEditable(false);

        mensaje2.setBounds(10,348,400,39);
        add(mensaje2);

        textarea = new JTextArea();
        scrollpanel = new JScrollPane(textarea);

        scrollpanel.setBounds(10,50, 400, 300);
        add(scrollpanel);

        salir.setBounds(420, 10, 100, 30);
        add(salir);

        textarea.setEditable(false);
        salir.addActionListener(this);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

    } // constructor
```

Se ha anulado el cierre de la ventana para que la finalización del servidor se haga desde el botón *Salir*. Cuando se pulsa el botón se cierra el **ServerSocket** y finaliza la ejecución:

```
    public void actionPerformed(ActionEvent e) {

        if(e.getSource() == salir) { // Pulsamos en el boton salir
            try {
                servidor.close(); // Cerramos el chat
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            System.exit(0);
        }

    } // actionPerformed
```

Desde el método *main()* se inicia el servidor y las variables y se prepara la pantalla:

```
    public static void main(String[] args) throws IOException {
```

```
servidor = new ServerSocket(PUERTO);
System.out.println("Servidor inciado...");
ServidorChat pantalla = new ServidorChat();
pantalla.setBounds(0, 0, 540, 450);
pantalla.setVisible(true);
mensaje.setText("Número de Conexiones Actuales: " + 0);
```

Se hace un bucle para controlar el número de conexiones. Dentro del bucle el servidor espera la conexión del cliente y cuando se conecta se crea el socket:

```
// El servidor admite hasta 10 conexiones
while (CONEXIONES < MAXIMO) {
    Socket socket = new Socket();
    try {
        socket = servidor.accept(); // Esperando un cliente
    } catch (SocketException ns) {
        //
        break; // Salimos del bucle
    }
```

El socket creado satisfactoriamente se almacena en la tabla, se cuenta el número de conexiones, se incremetan las conexiones actuales y se lanza el hilo para gestionar los mensajes del cliente que se acaba de conectar:

```
    tabla[CONEXIONES] = socket;
    CONEXIONES++;
    ACTUALES++;
    HiloServidor hilo = new HiloServidor(socket);
    hilo.start();
} // while
```

Se sale del bucle anterior si ha habido 10 conexiones o si se pulsa el botón *Salir*. Al pulsar el botón salir se cierra el *ServerSocket*, esto causa que la sentencia *socket = servidor.accept()* lance la excepción *SocketException* (ya que el servidor está cerrado) desde donde se hace *break* para salir del bucle.

Al salir del bucle se comprueba si el servidor está cerrado, si no lo está es que se han establecido las 10 conexiones, se visualiza un mensaje y se cierra el servidor:

```
// Cuando finaliza el bucle se cierra el servidor si no se ha cerrado antes
if (!servidor.isClosed())
    try {
        // Sale cuando se llega al maximo de conexiones
        mensaje2.setForeground(Color.red);
        mensaje2.setText("Maximo numero de conexiones establecidas: "
```

```
                                + CONEXIONES);
                servidor.close();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            System.out.println("Servidor finalizado ...");

    } // main

} // ServidorChat
```

El hilo **HiloServidor** se encarga de recibir y enviar los mensajes a los clientes del Chat. En el constructor, se recibe el socket creado y se crea el flujo de entrada desde el que se leen los mensajes que el cliente del Chat envía:

```
import java.io.*;
import java.net.*;

public class HiloServidor extends Thread {

    // Atributos
    DataInputStream fentrada;
    Socket socket = null;

    public HiloServidor(Socket s) {

        socket = s;
        try {
            // Se crea el flujo de entrada
            fentrada = new DataInputStream(socket.getInputStream());
        } catch (IOException e) {
            System.out.println("Error de E/S");
            e.printStackTrace();
        }

    } // constructor
```

En el método *run()*, lo primero que hacemos es enviar los mensajes que hay actualmente en el Chat al programa cliente para que los visualice en la pantalla. Esto se hace en el método *EnviarMensajes()* (se definirá más adelante). Los mensajes que se envían son los que están en el TextArea del Servidor de Chat:

```
    public void run() {

        ServidorChat.mensaje.setText("Numero de Conexiones Actuales: "
                                     + ServidorChat.ACTUALES);
        // Nada más conextarse el cliente le envio todos los mensajes
        String texto = ServidorChat.textarea.getText();
        EnviarMensajes(texto);
```

A continuación se hace un bucle while en el que se recibe lo que el cliente escribe en el chat. Cuando un cliente finaliza (pulsa el botón Salir de su pantalla) envía un asterisco al Servidor de Chat, entonces se sale del bucle while, ya que termina el proceso del cliente, de esta manera se controlan las conexiones actuales:

```
        while(true) {
            String cadena = "";
            try {
                cadena = fentrada.readUTF(); // Leemos lo que escribe el cliente
                // Cuando un cliente finaliza envia un *
                if (cadena.trim().equals("*")) {
                    ServidorChat.ACTUALES --;
                    ServidorChat.mensaje.setText("Numero de Conexiones Actuales: "
                                                 + ServidorChat.ACTUALES);
                    break; // Salir del while
                }
```

El texto que el cliente escribe en su chat, se añade al TextArea del servidor y el servidor enviará a todos los clientes el texto que hay en su TextArea llamando de nuevo a EnviarMensaje(), así todos ven la conversación:

```
                ServidorChat.textarea.append(cadena + "\n");
                texto = ServidorChat.textarea.getText();
                EnviarMensajes(texto); // Se envia el texto a todos los clientes

            } catch (Exception e) {
                e.printStackTrace();
                break;
            }
        } // while

    } // run

} // HiloServidor
```

El método *EnviarMensajes()* envía el texto del TextArea a todos los sockets que están en la tabla de sockets, de esta manera todos ven la conversación. Será necesario abrir un stream de escritura a cada socket y escribir el texto:

```
    // Envia los mensajes del TextArea a los clientes de chat
    private void EnviarMensajes(String texto) {

        // Recorremos la tabla de sockets para enviarles los mensajes
        for (int i = 0; i < ServidorChat.CONEXIONES; i++) {
            Socket s = ServidorChat.tabla[i];
            try {
                DataOutputStream fsalida = new DataOutputStream(s.getOutputStream());
                fsalida.writeUTF(texto);
            } catch (SocketException se) {
```

```
                // Esta excepcion ocurre cuando escribimos en un socket
                // de un cliente que ha finalizado
                se.printStackTrace();
            } catch(IOException e) {
                e.printStackTrace();
            }
        } // for

    } // EnviarMensajes

} // HiloServidor
```

Desde el Programa Cliente se raelizan las siguientes funciones:

- En primer lugar se pide el nombre o nick que el usuario utilizará en el chat, *Figura 2*.
- Se crea un socket al servidor de chat en el puerto pactado. Si todo va bien, el servidor asignará un hilo al cliente y se mostrará en la pantalla de chat del cliente la conversión que hay hasta el momento, *Figura 3*. Si no se puede establecer la conexión, se visualiza un mensaje de error.
- El cliente puede escribir sus mensajes y pulsar el botón Enviar, automáticamente su mensaje erá enviado a todos los clientes del chat.
- El botón Salir finaliza la conexión del cliente de chat.

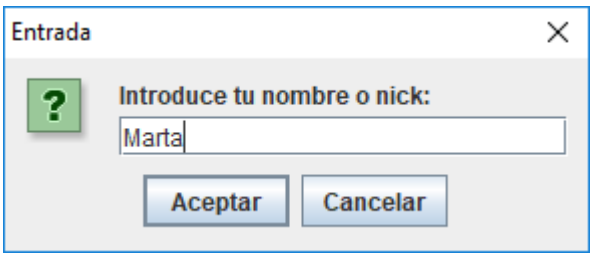

***Figura 2**. Pantalla para pedir el nombre o el nick*

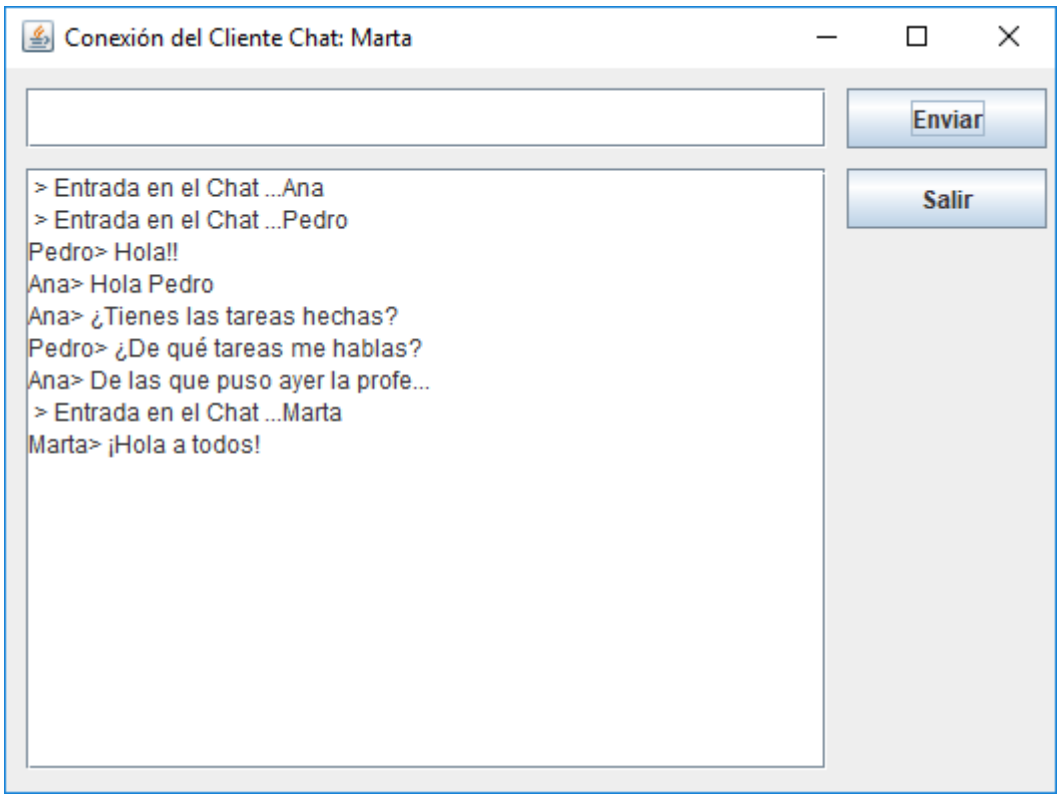

***Figura 3**. Pantalla del Cliente de Chat*

El código de la clase **ClienteChat** es el siguiente. En primer lugar se definen variables, campos de la pantalla y los streams de entrada y de salida:

```
import java.io.*;
import java.net.*;
import java.awt.event.*;
import javax.swing.*;

public class ClienteChat  extends JFrame implements ActionListener, Runnable {

    // Atributos
    private static final long serialVersionUID = 1L;
    Socket socket = null;

    //Streams
    DataInputStream fentrada; // Para leer los mensaje de todos
    DataOutputStream fsalida; // Para escribir los mensajes del cliente

    String nombre;
    static JTextField mensaje = new JTextField();
    private JScrollPane scrollpane1;
    static JTextArea textarea1;
    JButton boton = new JButton("Enviar");
    JButton desconectar = new JButton("Salir");
    boolean repetir  = true;
```

En el constructor se prepara la pantalla. Se escribe el socket creado y el nombre del Cliente de Chat:

```
// Constructor
public ClienteChat(Socket socket, String nombre) {

    super("Conexión del Cliente Chat: " + nombre);
    setLayout(null);

    mensaje.setBounds(10, 10, 400, 30);
    add(mensaje);

    textarea1 = new JTextArea();
    scrollpane1 = new JScrollPane(textarea1);
    scrollpane1.setBounds(10, 50, 400, 300);
    add(scrollpane1);

    boton.setBounds(420,  10,  100,  30);
    add(boton);

    desconectar.setBounds(420,  50,  100,  30);
    add(desconectar);

    textarea1.setEditable(false);
    boton.addActionListener(this);
    desconectar.addActionListener(this);

    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    this.socket = socket;
    this.nombre = nombre;
```

Se crean los flujos de entrada y salida. Se escribe en el flujo de salida un mensaje indicando que el usuario ha entrado en el chat. Este mensaje lo recibe el hilo (**HiloServidor**) y se lo manda a todos los clientes conectados:

```
    try {
        fentrada = new DataInputStream(socket.getInputStream());
        fsalida = new DataOutputStream(socket.getOutputStream());
        String texto = " > Entrada en el Chat ..." + nombre;
        fsalida.writeUTF(texto); // Escribe un mensaje de entrada
    } catch (IOException e) {
        System.out.println("Error de E/S");
        e.printStackTrace();
        System.exit(0);
    }

} // constructor
```

Cuando se pulsa el bóton *Enviar* se envía al flujo de salida el mensaje que el cliente ha escrito:

```
    // Definimos las acciones cuando pulsamos los botones
    public void actionPerformed(ActionEvent e) {
        // Se pulta el boton de Enviar
        if (e.getSource() == boton) {
            String texto = nombre + "> " + mensaje.getText();
            try {
                mensaje.setText("");
                fsalida.writeUTF(texto);
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }
```

Cuando se pulsa el bóton Salir se envía primero un mensaje indicando que el usuario abandona el chat y a continuación un asterisco indicando que el usuario va a salir del chat:

```
        // Se pulsa el boton de Desconectar
        if (e.getSource() == desconectar) {
            String texto = " > Abandona el Chat ... " + nombre;
            try {
                fsalida.writeUTF(texto);
                fsalida.writeUTF("*");
                repetir = false; // Para salir del bucle
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }

    } // actionPerformed
```

Dentro del método *run()*, el cliente lee lo que el hilo le manda (los mensajes de chat) para mostrarlo en el TextArea. Esto se realiza en un proceso repetitivo que termina cuando el usuario pulsa el botón *Salir*, que cambiará el valor de la variable *repetir* a *false* para que finalice el bucle:

```
    public void run() {
        String texto = "";

        while(repetir) {
            try {
                texto = fentrada.readUTF(); // Leer mensajes
                textarea1.setText(texto);   // Visualizar los mensajes
            } catch (IOException e) {
                // Este error sale cuando el servidor se cierra
                JOptionPane.showMessageDialog(null,  "Imposible conectar con el Servidor\n"
                                              + e.getMessage(), "<<Mensaje de Error:2>>",
                                              JOptionPane.ERROR_MESSAGE);
                repetir = false; // Salimos del bucle
            }
        } // while

        try {
```

```
            socket.close(); // Cerrar socket
            System.exit(0);
        } catch (IOException e) {
            e.printStackTrace();
        }

    } // ejecutar
```

En la función main() se pide el nombre del usuario, se realiza la conexión al servidor, se crea un objeto ClienteChat, se muestra la pantalla y se ejecuta el método run():

```
    public static void main(String[] args) {
        int puerto = 5050;
        String nombre = JOptionPane.showInputDialog("Introduce tu nombre o nick: ");
        Socket socket = null;

        try {
            // Cliente y servidor se ejecutan en la máquina local
            socket = new Socket("localhost", puerto);
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null,
                    "Imposible Conectar con el Servidor\n" + e.getMessage(),
                    "<<Mensaje de Error:1>>", JOptionPane.ERROR_MESSAGE);
            System.exit(0);
        }
        if (!nombre.trim().equals("")) { // Hay que escribir algo
            ClienteChat cliente = new ClienteChat(socket, nombre);
            cliente.setBounds(0, 0, 540, 400);
            cliente.setVisible(true);
            new Thread(cliente).start();
        } else {
            System.out.println("El nombre está vacio ...");
        }

    } // main

} // ClienteChat
```

Para ejecutar el servidor de chat se necesita que las clases java ServidorChat e HiloServidor estén en la misma carpeta. El programa cliente ClienteChat puede estar en cualquier otra carepta. Primero se ejecuta el programa servidor:

```
C:\> javac ServidorChat.java
C:\> javac HiloServidor.java

C:\> java ServidorChat
```

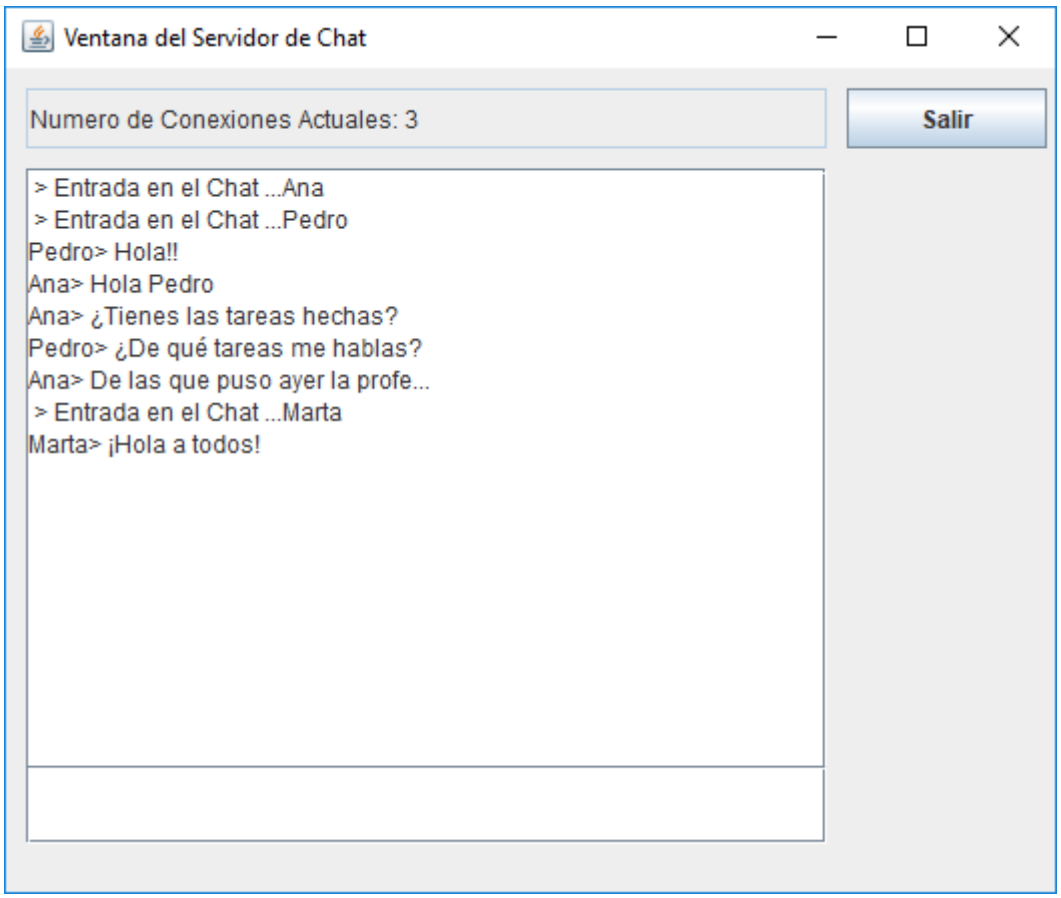

***Figura 4**. Ventana del Servidor de Chat*

Y luego el cliente desde la carpeta donde esté:

```
C:\> javac ClienteChat.java

C:\> java ClienteChat
```

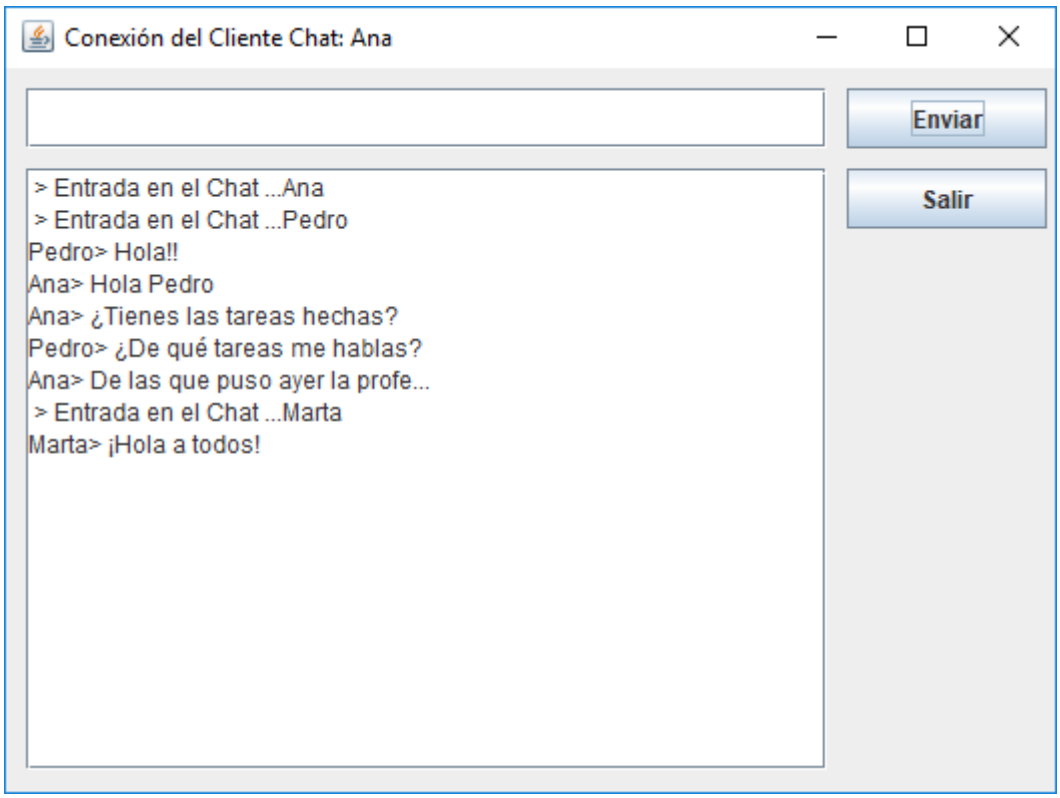

***Figura 5**. Ventana del cliente 1*

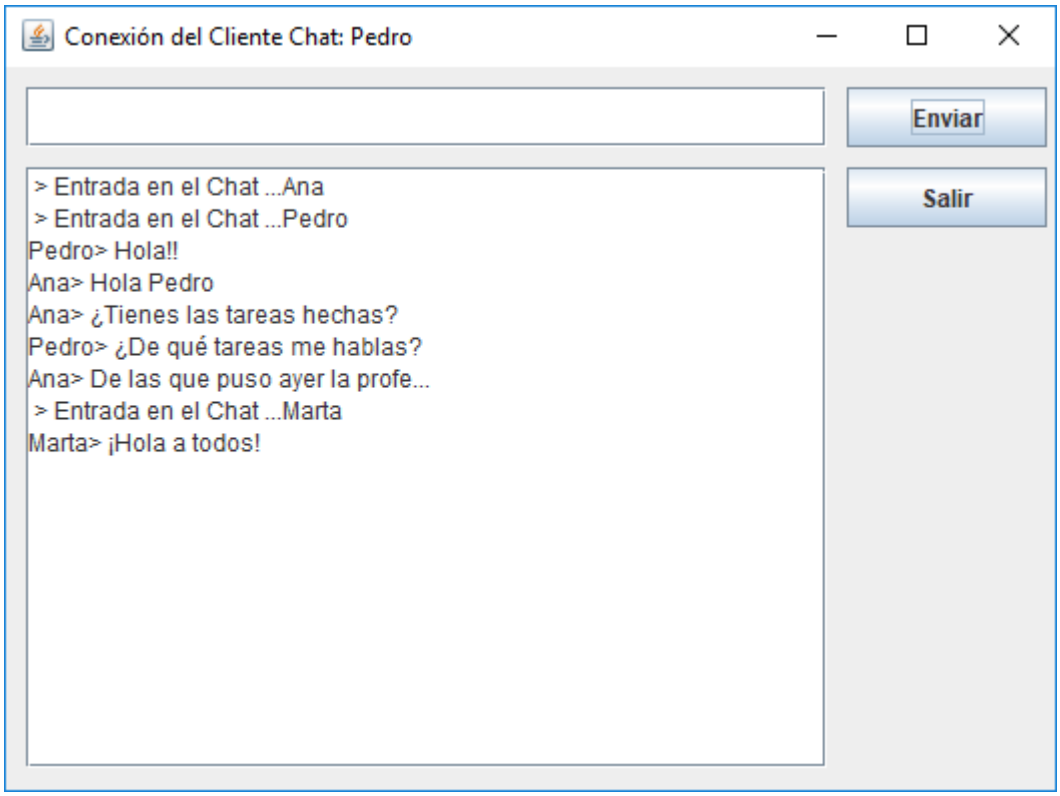

***Figura 6**. Ventana del cliente 2*

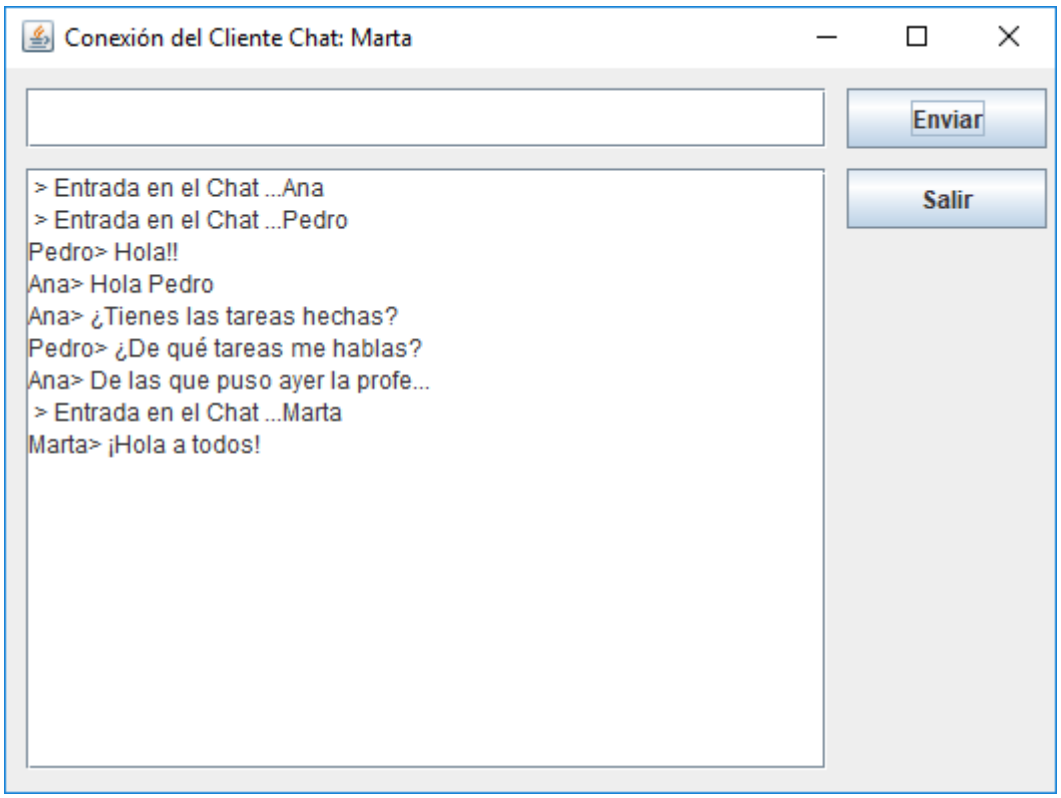

***Figura 7**. Ventana del cliente 3*

En el código expuesto el programa cliente y el servidor se ejecutan en la misma máquina. Pero lo normal es que el servidor esté en una maquina y el cliente en otra. En este caso es necesario especificar en el programa cliente, en la creación del socket, la dirección IP donde está el servidor de chat, por ejemplo si el servidor se ejecuta en la máquina con la IP 192.168.0.194, creamos el socket de la siguiente manera:

```
// El servidor admite hasta 10 conexiones
Socket socket = new Socket("192.168.0.194", puerto);
```

Se pide realizar la práctica anterior y comprobar que funciona correctamente, subir el código a la plataforma con una explicación del funcionamiento de la aplicación y de algún pantallazo tanto del programa servidor como del programa cliente.