PROGRAMACIÓN DE COMUNICACIONES EN RED.

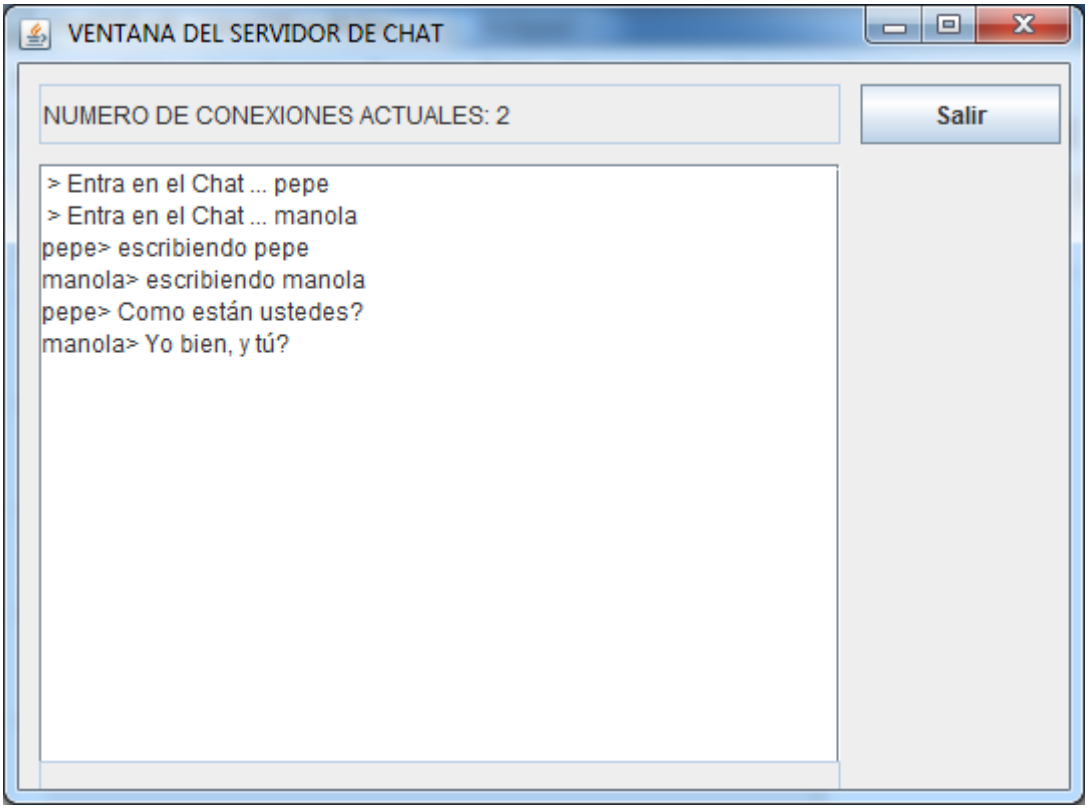
CHAT TCP.

Índice de contenido

1. CHAT TCP

Una situación típica de un servidor que atiende a múltiples clientes es un servidor de chat. Vamos a construir uno 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 siguiente:

- 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 en un textarea. El botón *Salir* finaliza el servidor de chat.
- 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 conversación del chat a todos los clientes conectados.
- Cuando se conecta un cliente se incrementa en un contador el número de conexiones actuales (ACTUALES), si se desconecta se decrementa. Otro contador (CONEXIONES) se usará para contar las conexiones de clientes, el máximo de conexiones viene dado en la variable MAXIMO.
- 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 admite hasta un máximo de conexiones, en el ejemplo 10.



El **programa servidor, ServidorChat,** es el siguiente. 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{
    private static final long serialVersionUID = 1L;
    static ServerSocket servidor;
    static final int PUERTO = 44444;// puerto por el que escucha
    static int CONEXIONES = 0; //cuenta las conexiones
    static int ACTUALES = 0;//nº de conexiones actuales activas
    static int MAXIMO=10; //máximo 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[10];//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, 30);add(mensaje2);
        mensaje2.setEditable(false) ;

        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);
        }
```

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:

```
// Acción cuando pulsamos botón Salir
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == salir) { // SE PULSA SALIR
    try {
        servidor.close(); //cierro
    }catch (IOException e1) {
    e1.printStackTrace();
    }
    System.exit(0);//fin
    }
}
```

Desde *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 iniciado ... ");
    ServidorChat pantalla = new ServidorChat();
    pantalla.setBounds(0, 0, 540, 400);
    pantalla.setVisible(true);
    mensaje.setText("NUMERO 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 un socket:

```
//SE ADMITEN HASTA 10 CONEXIONES
while (CONEXIONES < MAXIMO) {
    Socket s = new Socket() ;
    try {
        s = servidor.accept() ;// esperando cliente
    }catch (SocketException ns) {
    //sale por aqui si pulsamos botón Salir y
    //no se ejecuta todo el bucle
        break; //salir del bucle
    }
```

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

```
tabla [CONEXIONES] = s; //almacenar socket
CONEXIONES++;
ACTUALES++;
HiloServidor hilo = new HiloServidor(s);
hilo.start(); //lanzar hilo
}//fin 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 s = *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 bucle cerrar servidor si no se ha cerrado antes
if (!servidor.isClosed())
try {
    // sale cuando se llega al maXlmo de conexiones
    mensaje2.setForeground(Color.red);
    mensaje2.setText("MAXIMO N° DE CONEXIONES ESTABLECIDAS: "+ CONEXIONES);
    servidor.close();
}catch (IOException e1) {
    e1.printStackTrace();
}
System.out.println("Servidor finalizado ... ");
}//main

}//  Fin servidorChat
```

El hilo **HiloServidor** se encarga de recibir y enviar los mensajes a los clientes de 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 de chat envía:

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

public class HiloServidor extends Thread {
    DataInputStream fentrada;
    Socket socket = null;
    public HiloServidor(Socket s) {
        socket = s;
        try {
        // CREO FLUJO DE ENTRADA
        fentrada = new DataInputStream(socket.getInputStream());
        }catch (IOException e) {
            System.out.println("ERROR DE E/S");
            e.printStackTrace();
        }
    }
```

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().* 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 MAS CONECTARSE1EL 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();//lee lo que el cliente escribe
        //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 *EnviarMensajes(),,* así todos ven la conversación:

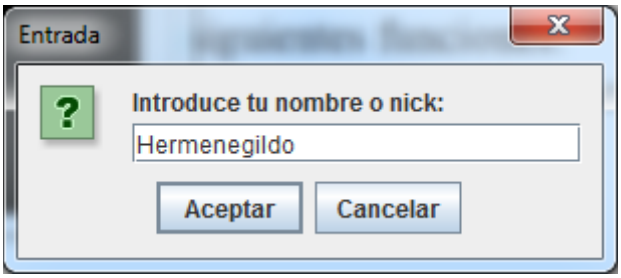
```
            ServidorChat.textarea.append(cadena + "\n");//añadir cadena al textarea del servidor
            texto = ServidorChat.textarea.getText();
            EnviarMensajes(texto);//envio texto a todos los clientes
        }catch (Exception e) {
            e.printStackTrace();
        break;
        }
    }// fin while
}//run
```

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:

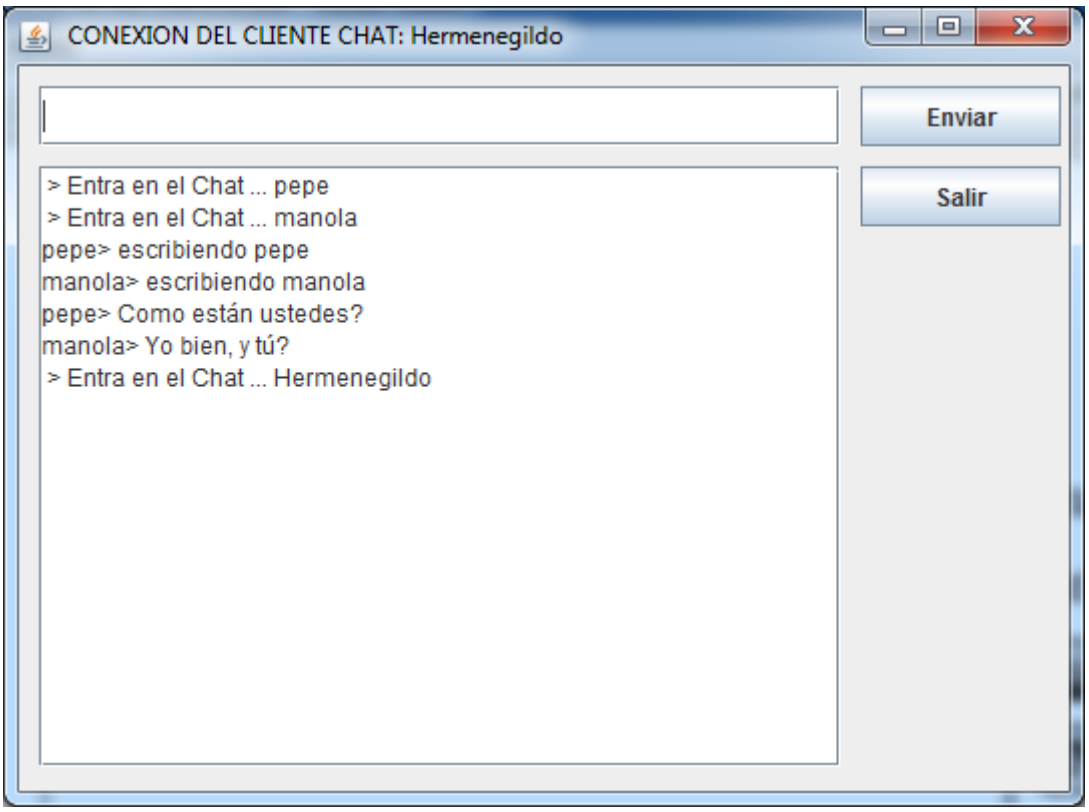
```
    // ENVIA LOS MENSAJES DEL TEXTAREA A LOS CLIENTES DEL CHAT
    private void EnviarMensajes(String texto) {
        int i;
        //recorremos tabla de sockets para enviarles los mensajes
        for (i = 0; i < ServidorChat.CONEXIONES; i++) {
            Socket s1 = ServidorChat.tabla[i];//obtener socket
            try {
                DataOutputStream fsalida = new DataOutputStream(s1.getOutputStream());
                fsalida.writeUTF(texto);//escribir en el socket el texto
            }catch(SocketException se){
                // esta excepción ocurre cuando escribimos en un socket
                // de un cliente que ha finalizado
            }catch (IOException e) {
                e.printStackTrace();
            }
        }// for
    }// EnviarMensajes
}// .. Fin HiloServidor
```

Desde el **programa cliente** se realizan las siguientes funciones:

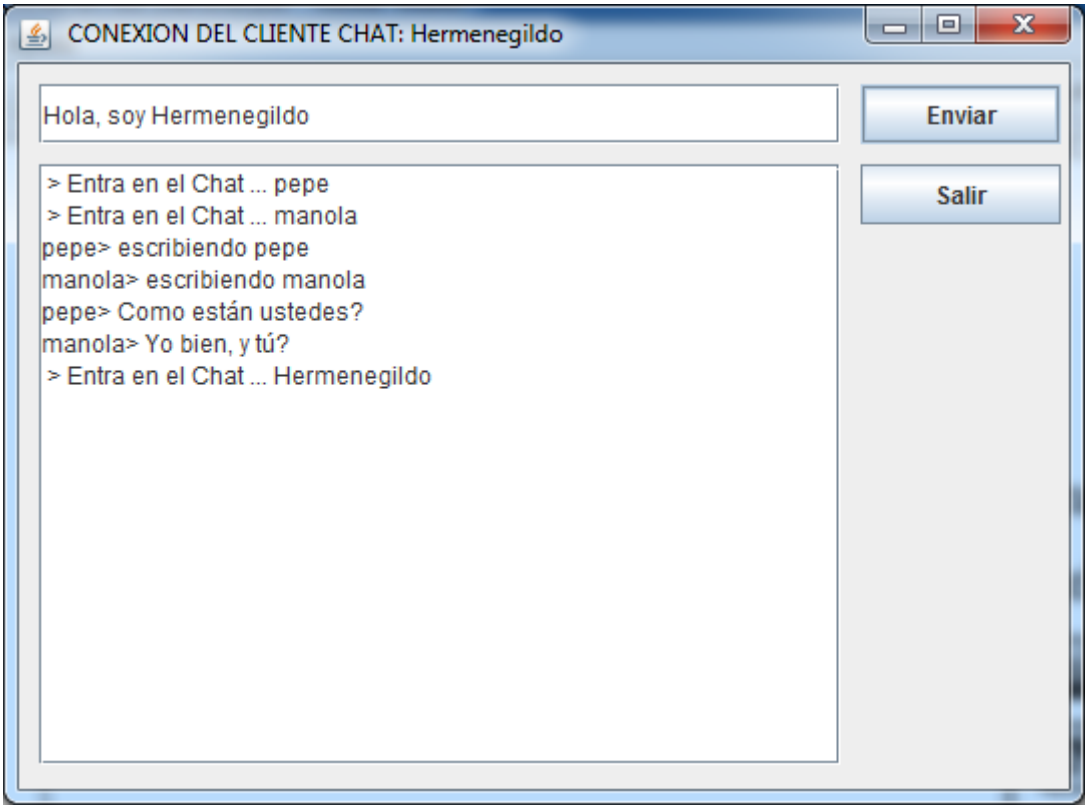
- En primer lugar se pide el nombre que el usuario utilizará en el chat.

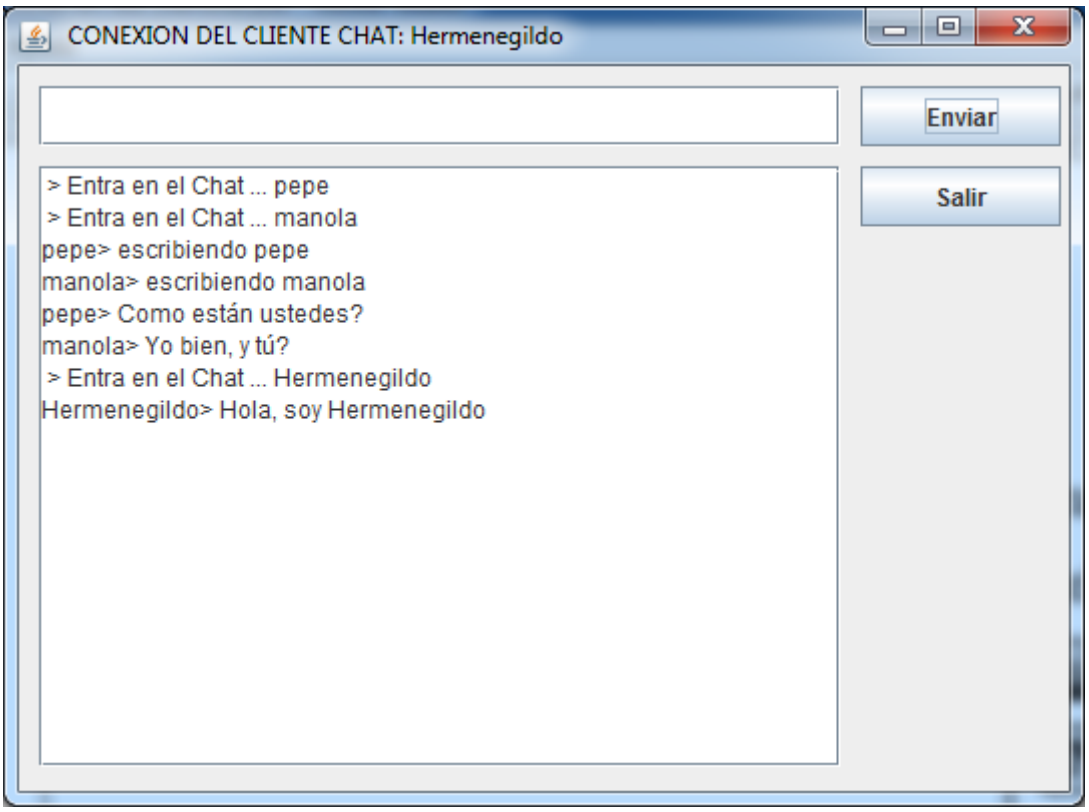


- Se crea un socket al servidor de chat en el puerto pactado. Si todo va bien, el servidor asignará un hilo al cliente y se mostrará en la pantalla de chat del cliente la conversación que hay hasta el momento. Si no se puede establecer la conexión, se visualiza un mensaje de error.



- El cliente puede escribir sus mensajes y pulsar el botón *Enviar,* automáticamente su mensaje será enviado a todos los clientes de chat.





- El botón *Salir* finaliza la conexión 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.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

public class ClienteChat extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;
    Socket socket = null;
    // streams
    DataInputStream fentrada; //para leer mensajes de todos
    DataOutputStream fsalida; //para escribir sus mensajes
    String nombre;
    static JTextField mensaje = new JTextField();
    private JScrollPane scrollpanel;
    static JTextArea textarea1;
    JButton boton = new JButton("Enviar");
    JButton desconectar = new JButton("Salir");
    boolean repetir = true;
```

En el constructor se prepara la pantalla. Se recibe el socket creado y el nombre del cliente de chat:

```
// constructor
public ClienteChat(Socket s, String nombre) {
    super(" CONEXION DEL CLIENTE CHAT: " + nombre);
    setLayout(null) ;
    mensaje.setBounds(10, 10, 400, 30);add(mensaje);
    textarea1 = new JTextArea();
    scrollpanel = new JScrollPane(textarea1);
    scrollpanel.setBounds(10, 50, 400, 300); add(scrollpanel);
    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);
    socket = s;
    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 (Hilo Servidor) y se lo manda a todos los clientes conectados:

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

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

```
//acción cuando pulsamos botones
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == boton) { // SE PULSA botón ENVIAR
        String texto = nombre + "> " + mensaje.getText();
        try {
            mensaje.setText("");//limpio area de mensaje
            fsalida.writeUTF(texto);
        }catch (IOException el) {
            el.printStackTrace();
        }
    }
```

Si se pulsa el botón *Salir* se envia primero un mensaje indicando que el usuario abandona el chat y a continuación un asterisco indicando que el usuario va a salir del chat:

```
            if (e.getSource() == desconectar) {// SE PULSA botón SALIR
                String texto = " > Abandona el Chat .,. " + nombre;
                try {
                    fsalida.writeUTF(texto);
                    fsalida.writeUTF("*");
                    repetir = false; //para salir del bucle
                }catch (IOException el) {
                    el.printStackTrace();
                }
            }
        }//
```

Dentro del método *ejecutar(),* 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 e false* para que finalice el bucle:

```
public void ejecutar() {
    String texto = "";
    while (repetir) {
        try {
            texto = fentrada.readUTF(); //leer mensajes
            textarea1.setText(texto); //visualizarlos
        }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; //salir 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 de usuario, se realiza la conexión al servidor, se crea un objeto **ClienteChat,** se muestra la pantalla y se ejecuta el método *ejecutan():*

```
    public static void main(String args[]) {
        int puerto = 44444;
        String nombre = JOptionPane.showInputDialog("Introduce tu nombre o nick:");
        Socket s = null;
        try {
            //cliente y servidor se ejecutan en la máquina local
            s=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(s, nombre);
            cliente.setBounds(0,0,540,400);
            cliente.setVisible(true);
            cliente.ejecutar();
        }else{
            System.out.println("El nombre está vacío");
        }
        }//main
}// .. ClienteChat
```

Aunque no se ha utilizado un hilo para implementar esta clase, lo más típico es usarlo; se implementaria **Runnable** para definir la clase:

public class **ClienteChat** extends Jframe implements ActionListener, **Runnable {**

Se cambiaría el método ejecutar por *run()* y se lanzaría el hilo cliente de la siguiente manera:

```
ClienteChat cliente = new ClienteChat (s, nombre);
cliente.setBounds(0, 0, 540, 400);
cliente.setVisible(true);
new Thread(cliente) .start() ;
```

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 carpeta.
Primero se ejecuta el programa servidor:

>java ServidorChat

Servidor iniciado ...

y luego el cliente desde la carpeta donde esté:

>java ClienteChat

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 máquina 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.