**Chat con TCP**

Servidor de Chat que atiende a múltiples clientes a la vez. Cada cliente será atendido en un hilo de ejecución. En el hilo se reciben los mensajes y se envían al resto.

El servidor muestra por pantalla el número de clientes que están conectados al chat y visualiza la conversación mantenida entre ellos. El botón Salir finaliza el servidor de chat.

El servidor se mantiene a la escucha de cualquier petición de nuevo cliente que se quiera conectar. El servidor acepta al cliente y guarda un array de sockets con las conexiones y que usará para enviar la conversación a todos los clientes conectados.

Cuando se conecta un cliente se incrementa en un contador el número de conexiones actuales, y disminuye si se desconecta.

**SOLUCION**

**ServidorChat.java**

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 se escucha**

static int CONEXIONES=0; // cuenta las conexiones

static int ACTUALES=0; // cuenta las conexiones 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

//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);

//SE ANULA EL CIERRE DE LA VENTANA para que la finalización del servidor

// se haga desde el botón Salir

setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

}

//Acción cuando se pulsa el botón Salir

**public void actionPerformed(ActionEvent e){**

if(e.getSource() == salir) { // Se pulsa salir

try {

**servidor.close(); // cierre**

} catch (IOException e1){

e1.printStackTrace();

}

System.exit(0); //fin

}

}

**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);

// BUCLE para controlar el número de conexiones.

// Desde el bucle se espera la conexión del cliente

// cuando hay una conexión se crea el Socket

while (CONEXIONES < MAXIMO) {

**Socket s= new Socket();**

try{

**s = servidor.accept();** // espera de cliente

} catch (SocketException ns){

// sale por aqui si se pulsa boton salir

// no se ejecuta por tanto el bucle

break; // sale del bucle

}

//almacenar el socket en la tabla y se incrementan las conexiones

// se lanza hilo para gestión de mensajes con el cliente nuevo

tabla[CONEXIONES]=s; //Almacenar el Socket

CONEXIONES++;

ACTUALES++;

**HiloServidor hilo=new HiloServidor(s);**

**hilo.start(); // lanzando hilo**

} // fin bucle

// cerrando el servidor si no se ha cerrado

if(!servidor.isClosed())

try{

// sale cuando llega al maximo de conexiones

mensaje2.setForeground(Color.red);;

mensaje2.setText("MAXIMO Nº DE CONEXIONES ESTABLECIDAS: " + CONEXIONES);

**servidor.close();**

} catch (IOException e1){

e1.printStackTrace();

}

System.out.println("Servidor finalizado...");

} //main

} // Fin Servidor Chat

**HiloServidor.java**

import java.io.\*;

import java.net.\*;

**public class HiloServidor extends Thread {**

**DataInputStream fentrada;**

**Socket socket =null;**

**public HiloServidor (Socket s) {**

**socket = s;**

try {

// crear flujo de entrada

**fentrada = new DataInputStream(socket.getInputStream());**

}catch(IOException e) {

System.out.println("ERROR DE E/S");

e.printStackTrace();

}

} // constructor

**public void run(){**

ServidorChat.mensaje.setText("NUMERO DE CONEXIONES ACTUALES: " + ServidorChat.ACTUALES);

// LO PRIMERO es enviar todos los mensajes

String texto=ServidorChat.textarea.getText();

EnviarMensajes(texto);

// bucle que recibe lo que el cliente escribe en el chat.

// al salir se le envía un \* al servidor

while (true) {

String cadena="";

try {

**cadena=fentrada.readUTF();** //lee lo que el cliente escribe

//cuando un cliente finaliza envía un \*

if(cadena.trim().equals("\*")) {

ServidorChat.ACTUALES--;

ServidorChat.mensaje.setText("NUMERO DE CONEXIONES ACTUALES: " + ServidorChat.ACTUALES);

break;//salir del while

} // fin if

ServidorChat.textarea.append(cadena + "\n");

texto = ServidorChat.textarea.getText();

**EnviarMensajes(texto);** // se envía texto a todos los clientes

} catch (Exception e) {

e.printStackTrace();

break;

}

} // fin bucle

} // fin run

// ENVIAR MENSAJES DEL TEXTAREA A LOS CLIENTES DEL CHAT

**private void EnviarMensajes(String texto) {**

int i;

// recorrer tabla de sockets para enviar mensajes a todos

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 excepcion ocurre al escribir en un socket

// de un cliente que ha finalizado

} catch (IOException e){ e.printStackTrace();}

} // fin for

}// fin Enviar mensajes

} // Fin HiloServidor

**ClienteChat.java**

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;**

// Preparar Streams

**DataInputStream fentrada;** // para mensajes de todos

**DataOutputStream fsalida;** // para escribir los mensajes

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;

// Constructor recibe el socket creado y el nombre del cliente

**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();

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);

**socket=s;**

this.nombre = nombre;

//crear flujos de E y S.

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

//al pulsar el boton enviar se envia al flujo de salida el mensaje escrito

//accion cuando se pulsan botones

**public void actionPerformed(ActionEvent e) {**

if (e.getSource() == boton) { //se pulsa el boton Enviar

String texto= nombre + " > " + mensaje.getText();

try{

mensaje.setText("");; // se limpia area de mensaje

fsalida.writeUTF(texto);

} catch (IOException e1) {

e1.printStackTrace();

}

} // fin if

// si se pulsa salir primero se envia mensaje de abandono de chat

if (e.getSource() == desconectar) { // se pulsa el boton Salir

String texto = " > Abandona el Chat ..." + nombre;

try {

**fsalida.writeUTF(texto);**

**fsalida.writeUTF("\*");**

repetir = false; // para salir del bucle

} catch (IOException e1) {

e1.printStackTrace();

}

}//fin if

} //fin action

//metodo ejecutar. el cliente lee lo que le envia el hilo y lo muestra en textarea

**public void ejecutar(){**

String texto = "";

while (repetir) {

try{

**texto = fentrada.readUTF();** // lee los mensajes

textarea1.setText(texto); //visualiza mensajes

} catch (IOException e) {

JOptionPane.showMessageDialog(null, "IMPOSIBLE CONECTAR CON EL SERVIDOR \n" + e.getMessage(),"<<MENSAJE DE ERROR:2>>",JOptionPane.ERROR\_MESSAGE);

repetir=false; // salir del bucle

}

} //fin while

try {

socket.close(); //cerrar socket

System.exit(0);

} catch (IOException e) {

e.printStackTrace();

}

} // fin ejecutar

// en main se pide nombre de usuario, se realiza la conexión y se crea el objeto clientechat

// arranca ejecutar

**public static void main(String[] args) {**

int puerto = 44444;

String nombre = JOptionPane.showInputDialog("Introduce tu Nick: ");

Socket s = null;

try {

**s = new Socket("localhost",puerto);**

} catch (IOException e){

JOptionPane.showMessageDialog(null, "IMPOSIBLE CONECTAR AL 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 esta vacio ...");

}

}// fin main

} // Cliente Chat