**Instituto Tecnológico de Tapachula**

Sistemas Distribuidos

MC en CC. Anamim Villarreal Wong

Chat por medio de Sockets en Java

Integrantes

Escalante Velázquez Fredy Rogelio De Jesús

Espinal Sibaja María Fernanda

Lozano Hilerio Alejandro

Ramírez López Diana Elisa

Vázquez Hernández Norma Karina

Contenido

[Desarrollo 3](#_Toc445286089)

[**Servidor** 3](#_Toc445286090)

[- Vista 3](#_Toc445286091)

[- Clase principal 4](#_Toc445286092)

[- Clase conectar 4](#_Toc445286093)

[**Cliente** 7](#_Toc445286094)

# Introducción

En el siguiente apartado se observa la elaboración de un chat, el cual se constituyó abriendo un puerto dentro la máquina, obteniendo la dirección ip para que permita que exista la comunicación entre dos máquinas.

Este consta de un servidor y un solo cliente, trabajando con un hilo “thread” para el proceso de comunicación entre el cliente y el servidor.

Esta práctica constituye a la aplicación de un sistema remoto, de manera que estas se encuentren cableadas.

# Descripción

El código de un chat consta de dos partes esenciales.

El servidor que es quien recibe el mensaje de un cliente, lo procesa y lo envía a otro cliente (senderización); esté proceso se efectúa mediante una interfaz donde el servidor visualiza los mensajes, la clase ***Messenger*** crea un objeto de la clase ***Conector*** y visualiza sus métodos, la clase ***Conector***  se extiende de Thread, lo que nos indica que todos los métodos establecidos en ella son parte de un hilo lo cuales son, el constructor de ella misma que nos permite hacer uso de todos sus métodos mediante un *super* , el método run hace la conexión del servidor y el cliente y pone a la variable “entrada” en modo escucha (percibiendo si llega algún mensaje), la método enviarMSG concatena los mensajes anteriores y el nuevo mensaje y los coloca en la variable “salida” para que el método run envié el mensaje al cliente y el método desconectar corta la conexión entre el servidor y el cliente para finalizar el proceso.

El cliente es quien envía mensajes para ser procesados por el servidor y poder recibir una respuesta, esté proceso se efectúa mediante de una interfaz gráfica donde el cliente debe de poner su dirección IP para poder conectarse con el servidor y así poder enviar y recibir mensajes, la clase ***Messenger*** crea un objeto de la clase Conector y visualiza sus métodos, la clase ***Conector*** se extiende de Thread, lo que nos indica que todos los métodos ahí declarados corren sobre el mismo hilo/proceso, los cuales son, el constructor de la clase en el cual genera la conexión del cliente hacia el servidor (una entrada y una salida usando UTF, que es un formato de codificación de caracteres Unicode), el método run pone la variable “entrada” en modo escucha (percibiendo si llega algún mensaje) y mostrar los mensajes en la interfaz, el método enviarMSG recibe de la interfaz una cadena de texto para enviarla al servidor y se concatena este mensaje.

# Desarrollo

# **Servidor**

## Vista

package messenger;

/\*\*

\*

\* @author MaferK

\*/

public class VServidor extends javax.swing.JFrame {

/\*\*

\* Creates new form VServidor

\*/

public VServidor() {

initComponents();

}

**private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)** {

Messenger.initServer();

//Se llama a la clase Messenger que tiene el método para inicializar el servidor

}

**private void jButton3ActionPerformed(java.awt.event.ActionEvent evt**) {

Messenger.servidor.enviarMSG(this.TXTmsgEnviar.getText());

//obtenemos el mensaje que vamos a enviar

}

**private void jButton4ActionPerformed(java.awt.event.ActionEvent evt)** {

Messenger.servidor.desconectar(); //se realiza para desconectar el servidor

}

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

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new VServidor().setVisible(true);

}

});

}

## Clase principal

package messenger;

/\*\*

\*

\* @author MaferK

\*/

public class Messenger {

public static Conector servidor,cliente;

public static void main (String[] args)

{

VServidor server = new VServidor(); //creación de la conexión del servidor lo sabemos //porque no tiene parámetros

server.show(); // mostramos la vista

}

public static void initServer()

{

//Se crea el método que tendrá como tarea iniciar el hilo.

servidor = new Conector("hilos");

servidor.start();

}

}

## Clase conectar

package messenger;

import java.net.\*;

import java.io.\*;

import messenger.VServidor;

/\*\*

\*

\* @author MaferK

\*/

public class Conector extends Thread {

private Socket s;

private ServerSocket ss;

private InputStreamReader entradaSocket;

private DataOutputStream salida;

private BufferedReader entrada;

final int puerto = 8180;

private String mensa="";

public Conector(String nombre)

{

super(nombre);

}

public void enviarMSG(String msg)

{

try{

salida.writeUTF("servidor "+msg+"\n"); //Es para leer cualquier tipo de caracter

VServidor.jTextArea1.setText(VServidor.jTextArea1.getText()+"\n"+"Servidor "+msg); VServidor.TXTmsgEnviar.setText("");

}catch (IOException e){};

}

public void run()

{

String text="test";

try{

ss = new ServerSocket(puerto);

s = ss.accept(); //Genera la conexion cliente-servidor

entradaSocket = new InputStreamReader(s.getInputStream()); // maneja la entrada de datos

entrada = new BufferedReader(entradaSocket); //Buffer para leer los datos de la entrada

salida = new DataOutputStream(s.getOutputStream()); //salida de datos

while(true)

{

//ciclo para leer mensajes para que funcione en paralelo con los demás componentes

text = entrada.readLine();

System.out.println(text);

VServidor.jTextArea1.setText(VServidor.jTextArea1.getText()+"\n"+text);

}

}catch (IOException e){

System.out.println("Algun Tipo de error");

};

}

public void desconectar()

{

//método para desconectar el servidor

try{

s.close();

}catch(IOException e){};

try{

ss.close();

}catch(IOException e){};

}

}

# **Cliente**

## Vista

package messenger;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author MaferK

\*/

public class VCliente extends javax.swing.JFrame {

public VCliente() {

initComponents();

}

**private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)** {

Messenger.initCliente(this.TXTip.getText()); //aquí se llama a la clase Messenger pasando la ip

}

**private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)** {

Messenger.cliente.enviarMSG(this.TXTenviarMSG.getText()); //

//Se llama a la Clase Messenger para pasaro los datos de entrada atraves del objeto cliente hacia //su clase conector

}

**private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {**

try {

Messenger.desconectarCliente();

System.exit(0);

} catch (InterruptedException ex) {

Logger.getLogger(VCliente.class.getName()).log(Level.SEVERE, null, ex);

}

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

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new VCliente().setVisible(true);

}

});

}

## Clase principal

package messenger;

/\*\*

\*

\* @author MaferK

\*/

public class Messenger {

public static Conector servidor,cliente;

public static void main (String[] args){

VCliente cliente = new VCliente();

cliente.show();

}

**public static void initCliente(String ip)**

{

cliente = new Conector(ip); //mando como parámetro la ip o en caso de ser local el Localhost

cliente.start(); // se inicia el hilo

}

**public static void desconectarCliente() throws InterruptedException**

{

Conector.interrupted(); //Se interrumpe el proceso

}

}

## Clase Conector

package messenger;

import java.net.\*;

import java.io.\*;

/\*\*

\*

\* @author MaferK

\*/

public class Conector extends Thread{

private Socket s;

private ServerSocket ss;

private InputStreamReader entradaSocket; //Entrada de flujo de vector

private DataOutputStream salida; //salida de datos

private BufferedReader entrada; //para poder leer los mensajes

final int puerto = 8180; //el puerto es el que yo decida

public Conector(String ip) //se maneja el constructor y obtenemos la IP a la que nos //conectaremos

{

try{

s = new Socket(ip,puerto); //pasamos al socket el puerto y la ip

entradaSocket = new InputStreamReader(s.getInputStream()); //Dar de alta la entrada de //datos

entrada = new BufferedReader(entradaSocket); //Buffer para leer los datos de la entrada

salida= new DataOutputStream(s.getOutputStream()); //Creación de salida de datos para el //envío de mensajes

salida.writeUTF("Conectado- \n"); //lee cualquier tipo de caracteres

}catch (Exception e){};

}

public void run()

{

String texto; //guardo todo lo que recibo del servidor

while(true)

{try{

texto = entrada.readLine(); //Solo considera el término de línea hasta que encontramos un //salto

VCliente.jTextArea1.setText(VCliente.jTextArea1.getText()+"\n"+ texto);

}catch(IOException e){};

}

}

public void enviarMSG(String msg)

{

System.out.println("enviado");

try{

salida = new DataOutputStream(s.getOutputStream());

salida.writeUTF("Fer :"+msg+"\n"); //Es para leer cualquier tipo de caracter y concateno el nuevo mensaje

VCliente.jTextArea1.setText(VCliente.jTextArea1.getText()+"\n"+"fer :" +msg); VCliente.TXTenviarMSG.setText("");

}catch (IOException e){

System.out.println("Problema al enviar");

};

}

}

# Conclusión

En la práctica de la realización del chat comprendimos el uso del Buffer como la forma de la realización de la lectura de mensajes que son recibidos.

También se implementó una clase hilo en cliente y en servidor para la realización de todos los procesos que se requieren en el chat. La implementación de solo un hilo se debe a que solo se tiene un cliente.

En la realización del chat se tuvieron problemas con las conexiones entre el cliente y servidor, debido a las opciones de seguridad de las computadoras; por lo cual se desactivaron los firewall de ambas máquinas (cliente y servidor).

Dentro del código fue el reconocimiento de la IP, debido que no lo reconocía como parámetro al igual que con los puertos.