**PRACTICA DE TELEMATICA**

**Análisis, Diseño e Implementación de un chat multiusuario y temático** **utilizando Sockets**

**JULIAN ANDRES RODRIGUEZ RESTREPO**

**201110063010**

**CARLOS ANDRES CASTRO**

**Profesor**

**UNIVERSIDAD EAFIT**

**2013**

**INTRODUCCION**

Esta practica de la materia de telemática se desarrollo con el fin de aprender a utilizar el manejo de los puertos, por medio de la herramienta que el lenguaje java permitia, estos eran los sockets.

**DESARROLLO**

El chat que desarrolle maneja el lenguaje de programación java, y el chat esta compuesto por dos clases principalmente. La primera clase es la clase servidor, esta clase que a continuación será especificada punto a punto, es la clase que maneja cual es el puerto por el que un usuario con una IP debe entrar par que se establezca la conexión, al igual es la clase encarga de saber llevar los mensajes de un usuario a los demás usuarios que estén conectados en el chat.

Codigo:

- Clase Servidor

package servidor;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintStream;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Enumeration;

import java.util.List;

import java.util.Vector;

public class Servidor extends Thread

{

private static Vector clientes;

private Socket conexion;

private String nombreCliente;

private static List listaDeNombres = new ArrayList();

public Servidor(Socket socket)

{

this.conexion = socket;

}

public boolean almacenar(String nombre)

{

for (int i=0; i< listaDeNombres.size(); i++)

{

if(listaDeNombres.get(i).equals(nombre))

return true;

}

listaDeNombres.add(nombre);

return false;

}

public void remove(String oldName)

{

for (int i=0; i< listaDeNombres.size(); i++)

{

if(listaDeNombres.get(i).equals(oldName))

listaDeNombres.remove(oldName);

}

}

public static void main(String args[])

{

clientes = new Vector();

try

{

ServerSocket server = new ServerSocket(5554);

System.out.println("Servidor en el socket 5554");

while (true)

{

Socket conexion = server.accept();

Thread t = new Servidor(conexion);

t.start();

}

}

catch (IOException e)

{

System.out.println("IOException: " + e);

}

}

public void run()

{

try

{

BufferedReader entrada = new BufferedReader(new InputStreamReader(this.conexion.getInputStream()));

PrintStream salida = new PrintStream(this.conexion.getOutputStream());

this.nombreCliente = entrada.readLine();

if (almacenar(this.nombreCliente))

{

salida.println("Nombre ya existente");

clientes.add(salida);

this.conexion.close();

return;

}

else

{

System.out.println(this.nombreCliente + " Se ha conectado");

}

if (this.nombreCliente == null)

{

return;

}

clientes.add(salida);

String msg = entrada.readLine();

while (msg != null && !(msg.trim().equals("")))

{

sendToAll(salida, " Escribe :: ", msg);

msg = entrada.readLine();

}

System.out.println(this.nombreCliente + " Conexion Terminada ");

sendToAll(salida, " Se ha desconectado", " ");

remove(this.nombreCliente);

clientes.remove(salida);

this.conexion.close();

}

catch (IOException e)

{

System.out.println("Fallo de conexion"+" IOException: " + e);

}

}

public void sendToAll(PrintStream salida, String a, String msg) throws IOException

{

Enumeration e = clientes.elements();

while (e.hasMoreElements())

{

PrintStream chat = (PrintStream) e.nextElement();

if (chat != salida)

{

chat.println(this.nombreCliente + a + msg);

}

}

}

}

La clase servidor se extiende a implementar hilos se crean 4 variables locales las cuales son: clientes que es de tipo vector, conexión que es de tipo socket, nombreCliente que es de tipo string y listaDeNombres que es de tipo arraylist.

El constructor inicializa la variable conexión que es un tipo de socket y creamos un método de tipo booleano que nos permite almacenar los nombres siempre y cuando estos no estén repetidos, y la clase remove que es la que elimina el nombre de los usuarios que se han desconectado.

La clase main inicializa la variable vector cliente que es encargada de almacenar datos que un cliente ingrese, y método main aparte de inicializar la variable cliente se encarga de abrir un socket con su respectivo identificador, en el caso del código, el puerto o socket a utilizar es el 5554, y se abre un while que si todo se inicializo correctamente se creara un hilo que es el q iniciara el servidor.

La clase run es la encargada de hacer que el servidor cada vez que tenga peticiones de mensajes o conexiones entrantes o salientes actualize el estado del servidor y comienza inicializando entrada de tipo bufferreader, salida de tipo printstream, y a la variable nombreCliente le asigna una entrada, la cual va a ser una línea de mensaje. El método comienza también evaluando el nombre del cliente, este debe meter un nombre que no este siendo usado por otro, de lo contrario el programa terminara conexión, en el debido caso de que el programa termine una conexión este se cierra, si el nombre del usuario es correcto, inmediatamente imprime dentro del servidor (para decirle a quien controla el servidor, quien se desconecto o conecto en su debido momento) que tal usuario se ha conectado, en el caso de que el nombre este vacio o nulo, le envía un mensaje que no es un usuario valido. De ahí en adelante, si llega un mensaje de un cliente y si este mensaje en diferente de nulo o diferente de vacio, inicializa el método sendToAll, el cual imprime el mensaje que llego al servidor a todos lo usuarios que estén conectados al servidor.

* Clase Cliente

package cliente;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintStream;

import java.net.Socket;

import java.util.Scanner;

public class Cliente extends Thread {

private Socket conexion;

public Cliente(Socket socket)

{

this.conexion = socket;

}

public static void main(String args[])

{

try {

System.out.println("Digite la ip del servidor");

Scanner leer = new Scanner(System.in);

String ip = leer.next();

Socket socket = new Socket(ip , 5554);

PrintStream salida = new PrintStream(socket.getOutputStream());

BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Digite su nombre: ");

String nombre = teclado.readLine();

salida.println(nombre.toUpperCase());

Thread thread = new Cliente(socket);

thread.start();

String msg;

while (true)

{

System.out.print("Mensaje > ");

msg = teclado.readLine();

salida.println(msg);

}

}

catch (IOException e)

{

System.out.println("Fallo de conexion" + " IOException: " + e);

}

}

public void run()

{

try {

BufferedReader entrada = new BufferedReader(new InputStreamReader(this.conexion.getInputStream()));

String msg;

while (true)

{

msg = entrada.readLine();

if (msg == null) {

System.out.println("Conexion terminada");

System.exit(0);

}

System.out.println();

System.out.println(msg);

System.out.print("Responder > ");

}

} catch (IOException e) {

System.out.println("Fallo en el envio del mensaje" + " IOException: " + e);

}

}

}

Este comienza inicializando una variable privada conexion de tipo socket y en el constructor inicializa el socket. En el método main lo que se hace es pedir primero la ip del servidor digitada por el usuario y se la va a entregar al socket que es una dupla de la ip que el usuario registra y en un puerto que es que esta definido en el servidor (en este caso 5554) si todo sale bien, le pedirá que ingrese el usuario, como vimos anteriormente, si el usuario es correcto, este se conectara al servidor, de lo contrario el servidor lanzara un mensaje diciendo que el usuario no es correcto o existente. Al entrar un usuario valido, se activa la conexión y se crea un hilo, y aparece un while, en el cual si no nos ha llegado un mensaje, nos imprimirá un “mensaje >” y leera línea de texto que el usuario ingrese, y en el mensaje ingresado será enviado al servidor, para que este cuando lo reciba, se lo envie a todos los usuarios que estén conectados.

El método run lo que hace básicamente es si se entra un nulo como mensaje, quiere decir que la conexión debe terminar y cierra la conexión de lo contrario si llega un mensaje de otro usuario lo que hace es imprime el mensaje y a continuación imprime “Responder >” y asi sucesivamente hasta que el usuario se desconecte.

**INSTALACION**

Para poder correr el programa, se debe estar actualizado preferiblemente en la versión 7u21 del java development kit o java normal, de lo contrario este enviara error y no permitirá inicializar ni el servidor ni el chat.

Despues de revisar lo anterior debemos inicializar primeramente el servidor para eso hacemos lo siguiente:

En Windows:

* Abrimos la consola
* Vamos a la ruta donde se encuentra nuestro servidor.jar
* Ejecutamos el siguiente comando: java –jar servidor.jar

En Mac o Linux:

* Abrimos la consola
* Vamos a la ruta donde se encuentra nuestro servidor.jar
* Ejecutamos el siguiente comando: java –jar servidor.jar

Si todo lo anterior estuvo correcto, nos debe imprimir en consola, que el servidor inicio en el socket 5554. Despues de hacer lo anterior inicializamos el chat, para eso hacemos lo siguiente:

En Windows:

* Abrimos la consola
* Vamos a la ruta donde se encuentra nuestro chat.jar
* Ejecutamos el siguiente comando: java –jar chat.jar

En Mac o Linux:

* Abrimos la consola
* Vamos a la ruta donde se encuentra nuestro chat.jar
* Ejecutamos el siguiente comando: java –jar chat.jar

Si todo lo anterior fue correcto nos debe pedir que ingresemos la ip del servidor, para ello si es en Windows, abrimos el cmd y tecleamos ipconfig, esto nos mostrara los parámetros de configuración de la red, allí buscamos la dirección ip, la cual deberá tener nuestro cliente, al cliente colocar la dirección ip, le pide que ingrese un nombre, el cliente se loguea con el nombre y empezara a disfrutar la charla con sus amigos.

**BIBLIOGRAFIA**

* [**http://codigoprogramacion.com/cursos/java/103-sockets-en-java-con-cliente-y-servidor.html#.UaQk9pXCD7Y**](http://codigoprogramacion.com/cursos/java/103-sockets-en-java-con-cliente-y-servidor.html#.UaQk9pXCD7Y)
* <http://www.youtube.com/watch?v=Uo5DY546rKY>
* Presentación java sockets.pdf enviada por el monitor Miguel Baquero