

**Tema**

SOCKETS JAVA

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

17/07/2023

DOMINIOS DE SEGURIDAD EN APLICACIONES JEE

[1. MARCO TEÓRICO 4](#_Toc121093428)

[1.1. OBJETIVOS 4](#_Toc121093429)

[1.2.1. OBJETIVO GENERAL 4](#_Toc121093430)

[1.2.2. OBJETIVOS ESPECÍFICOS 4](#_Toc121093431)

[1.2. JAVA. 4](#_Toc121093432)

[1.3. APACHE NETBEANS 4](#_Toc121093433)

[1.4. CONCURRENCIA 5](#_Toc121093434)

[1.5. HILOS 5](#_Toc121093435)

[1.6. HILOS PARALELOS Y CONCURRENTES 5](#_Toc121093436)

[1.7. TERMINOLOGÍA Y METODOS DE HILOS EN JAVA. 5](#_Toc121093437)

[1.8. PROTOCOLOS TCP/UDP. 6](#_Toc121093438)

[1.9. SOCKETS. 7](#_Toc121093439)

[2. PARTE PRÁCTICA 9](#_Toc121093440)

[2.1. SOCKET SERVIDOR 9](#_Toc121093441)

[2.1.1 CREACIÓN DEL PROYECTO. 9](#_Toc121093442)

[2.1.2 Creación de paquetes MVC. 10](#_Toc121093443)

[2.1.3 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 12](#_Toc121093444)

[2.1.4. CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 19](#_Toc121093445)

[2.1.5. CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 21](#_Toc121093446)

[2.1.6. EJECUCIÓN DEL PROYECTO. 36](#_Toc121093447)

[2.2. SOCKET CLIENTE 38](#_Toc121093448)

[2.1.1 CREACIÓN DEL PROYECTO. 38](#_Toc121093449)

[2.1.4 CREACIÓN DE CARPETAS PARA IMÁGENES Y AUDIO 40](#_Toc121093450)

[2.1.5 Creación de paquetes MVC. 41](#_Toc121093451)

[2.1.6 CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA 42](#_Toc121093452)

[2.1.7. CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO 66](#_Toc121093453)

[2.1.8. CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR 71](#_Toc121093454)

[2.1.9. EJECUCIÓN DEL PROYECTO. 84](#_Toc121093455)

[3. CONCLUSIONES 86](#_Toc121093456)

[4. RECOMENDACIONES 87](#_Toc121093457)

[5. BIBLIOGRAFIA 87](#_Toc121093458)

**INDICE DE IMÁGENES**

[Figura 1. Logotipo identificativo de Java 4](#_Toc121178762)

[Figura 2. Funcionamiento TCP 7](#_Toc121178763)

[Figura 3. Funcionamiento UDP 7](#_Toc121178764)

[Figura 4. Funcionamiento de una conexión Socket 8](#_Toc121178765)

[Figura 5. Logotipo identificativo de Netbeans 9](#_Toc121178766)

[Figura 6. Selección del tipo de aplicación a desarrollar 9](#_Toc121178767)

[Figura 7. Nombre del proyecto 10](#_Toc121178768)

[Figura 8. Estructura del proyecto generado 10](#_Toc121178769)

[Figura 9. Creación de paquete. 11](#_Toc121178770)

[Figura 10. Selección del nombre de los paquetes 11](#_Toc121178771)

[Figura 11. Eliminación del paquete main. 12](#_Toc121178772)

[Figura 12. Creación de clase Java. 12](#_Toc121178773)

[Figura 13. Codificación archivo FormularioPrincipal.java 12](#_Toc121178774)

[Figura 14. Imagen del servidor 13](#_Toc121178775)

[Figura 15. Diseño del FormularioPrincipal.java 13](#_Toc121178776)

[Figura 16. Creación de clase Java. 19](#_Toc121178777)

[Figura 17. Codificación archivo Cliente.java. 19](#_Toc121178778)

[Figura 18. Creación de clase Java. 21](#_Toc121178779)

[Figura 19.Codificación archivo ClientesControlador. 21](#_Toc121178780)

[Figura 20. Creación de la clase 24](#_Toc121178781)

[Figura 21. Codificación archivo SocketsControlador 24](#_Toc121178782)

[Figura 22. Creación de la clase 33](#_Toc121178783)

[Figura 23. Codificación archivo ServidorControlador 33](#_Toc121178784)

[Figura 24. Ejecución del proyecto 36](#_Toc121178785)

[Figura 25. Resultado de la ejecución 36](#_Toc121178786)

[Figura 26. Inicio del servidor 37](#_Toc121178787)

[Figura 27. Logotipo identificativo de Netbeans 38](#_Toc121178788)

[Figura 28. Selección del tipo de aplicación a desarrollar 38](#_Toc121178789)

[Figura 29. Nombre del proyecto 39](#_Toc121178790)

[Figura 30. Estructura del proyecto generado. 39](#_Toc121178791)

[Figura 31. Carpeta de imágenes 40](#_Toc121178792)

[Figura 32. Carpeta de sonido 40](#_Toc121178793)

[Figura 33. Creación de paquete. 41](#_Toc121178794)

[Figura 34. Selección del nombre de los paquetes 41](#_Toc121178795)

[Figura 35. Eliminación del paquete main 42](#_Toc121178796)

[Figura 36. Creación de clase Java 42](#_Toc121178797)

[Figura 37. Codificación archivo PantallaPrincipal 42](#_Toc121178798)

[Figura 38. Diseño del formulario de PantallaPrincipal.java 43](#_Toc121178799)

[Figura 39. Creación de clase Java. 53](#_Toc121178800)

[Figura 40. Codificación archivo FormularioLogin 53](#_Toc121178801)

[Figura 41. Diseño del formulario de FormularioLogin.java 54](#_Toc121178802)

[Figura 42. Creación de clase Java. 56](#_Toc121178803)

[Figura 43. Codificación archivo FormularioEnviarArchivos 56](#_Toc121178804)

[Figura 44. Diseño del formulario de FormularioEnviarArchivos.java 57](#_Toc121178805)

[Figura 45. Creación de clase Java 66](#_Toc121178806)

[Figura 46. Codificación archivo Estilos 66](#_Toc121178807)

[Figura 47. Creación de clase Java. 68](#_Toc121178808)

[Figura 48. Codificación archivo Sonidos.java 68](#_Toc121178809)

[Figura 49. Creación de clase Java. 71](#_Toc121178810)

[Figura 50. Codificación archivo EnviarArchivosControlador 71](#_Toc121178811)

[Figura 51. Creación de clase Java. 75](#_Toc121178812)

[Figura 52. Codificación archivo RecibirArchivosControlador 75](#_Toc121178813)

[Figura 51. Creación de clase Java. 79](#_Toc121178814)

[Figura 52. Codificación archivo MensajesControlador 79](#_Toc121178815)

[Figura 55. Ejecución del Servidor. 84](#_Toc121178816)

[Figura 56. Ejecución del Cliente. 85](#_Toc121178817)

[Figura 57. Ejecución del chat del cliente. 85](#_Toc121178818)

[Figura 58. Ejecución de varios clientes. 86](#_Toc121178819)

[Figura 59. Ejecución del cliente en diferentes computadoras. 86](#_Toc121178820)

[Figura 60. Envío de archivos. 86](#_Toc121178821)

**ÍNDICE DE TABLAS**

[Tabla 1. Terminología utilizada en Hilos Java 5](#_Toc121093518)

[Tabla 2. Código de FormularioPrincipal.java 14](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093519)

[Tabla 3. Codificación del archivo Cliente.java 20](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093520)

[Tabla 4. Codificación del archivo ClientesControlador 22](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093521)

[Tabla 5. Codificación del archivo SocketsControlador 25](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093522)

[Tabla 6. Codificación del archivo ServidorControlador 34](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093523)

[Tabla 7. Código del archivo PantallaPrincipal. 44](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093524)

[Tabla 8. Código del archivo FormularioLogin 55](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093525)

[Tabla 9. Código del archivo FormularioEnviarArchivos 58](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093526)

[Tabla 10. Codificación del archivo Estilos 67](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093527)

[Tabla 11. Codificación del archivo Sonidos.java 69](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093528)

[Tabla 12. Codificación del archivo EnviarArchivosControlador 72](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093529)

[Tabla 13. Codificación del archivo RecibirArchivosControlador 76](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093530)

[Tabla 14. Codificación del archivo MensajesControlador 80](file:///F:\7mo%20Semestre\Distribuidas\Sockets_Grupo6%20(1).docx#_Toc121093531)

# MARCO TEÓRICO

## OBJETIVOS

### OBJETIVO GENERAL

Comprender la funcionalidad de los hilos en Java para ejecutar multiples tareas al mismo así como también ver la manera en la que un programa realizado con hilos actúa de manera diferente a uno que no los utiliza.

### OBJETIVOS ESPECÍFICOS

* Analizar le proceso de los hilos en java.
* Conocer la forma de utilizar los hilos en el lenguaje de programación de java.
* Conocer la forma de ejecutar múltiples hilos al mismo tiempo y a su vez detenerlos.

## JAVA.

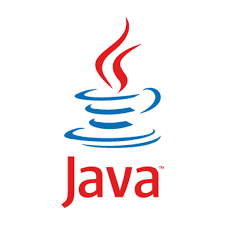


Figura 1. Logotipo identificativo de Java

Java es según la definición dada por [1] es un lenguaje de programación ampliamente utilizado para codificar aplicaciones de distintos indoles, desde aplicaciones de escritorio hasta tipo web, pasando por desarrollo de videojuegos, macrodatos, IA e IoT. Actualmente la ultima versión numero 8 y cuenta con 3 versiones disponibles de Java las cuales son Java SE, Java EE y Java ME.

La mayor fortaleza de java es su capacidad de ser un lenguaje sumamente apto para ser aplicado en el paradigma orientado a objetos, lo cual es su mayor fortaleza en este aspecto.

## APACHE NETBEANS

Apache netbeans es [2] un entorno de desarrollo integrado (IDE) creado para principalmente el lenguaje de programación de Java, anteriormente conocido simplemente como netbeans, cambio su nombre a apache netbeans y actualmente se encuentra en la versión 15.

Este IDE ofrece una serie de plugins que pueden ser instalados para facilitar el trabajo al momento de desarrollar aplicaciones y ejecutarlas, como por ejemplo poder instalar de manera rápida y fácil el servidor de payara o glassfish.

Cabe recalcar que apache netbeans es completamente gratis para su uso tanto para fines empresariales como educativos, y que se puede encontrar fácilmente para su descarga en el sitio oficial, teniendo versiones para los sistemas operativos Windows, Mac, Linux y Solaris.

## CONCURRENCIA

La concurrencia es [3] la ejecución de varios procesos a la vez, es decir, es la ejecución simultánea de múltiples tareas interactivamente. Estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por un único programa. Las tareas se pueden ejecutar en una sola CPU (multiprogramación), en varios procesadores, o en una red de computadores distribuidos.

## HILOS

Un hilo es [3] un flujo de control dentro de un programa. Creando varios hilos podremos realizar varias tareas simultáneamente. Cada hilo tendrá sólo un contexto de ejecución (contador de programa, pila de ejecución). Es decir, a diferencia de los procesos UNIX, no tienen su propio espacio de memoria, sino que acceden todos al mismo espacio de memoria común, por lo que será importante su sincronización cuando tengamos varios hilos accediendo a los mismos objetos.

Los hilos son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea de forma independiente. Por ejemplo un hilo puede encargarse de la comunicación con el usuario, mientras que otros actúan en segundo plano, realizando la transmisión de un fichero, accediendo a recursos del sistema (cargar sonidos, leer ficheros ...), etc.

## HILOS PARALELOS Y CONCURRENTES

Un hilo como se indicó con anterioridad se utiliza sobre todo para dividir tareas y hacerla de manera más eficiente, además como indica [4] permitiendo que estas se ejecuten de manera simultánea en vez de terminar una para continuar otra.

De manera más formal [5] “cada hilo progresa de forma independiente al resto. Esto provoca que cada uno de los hilos pueda potencialmente viajar a una velocidad distinta, ejecutándose "concurrentemente" con el resto. Esto provoca que la ejecución de un programa sea diferente en cada da pasada y que sea independiente.”

## TERMINOLOGÍA Y METODOS DE HILOS EN JAVA.

Antes de realizar la practica presente es necesario conocer algunos términos que son necesarios para entender los códigos y algunas definiciones que se utilizaran durante la práctica.

Tabla 1. Terminología utilizada en Hilos Java

|  |  |
| --- | --- |
| TÉRMINO O METODO | DESCRIPCIÓN |
| isInterrupted() | Método que verifica si un hilo esta interrumpido o no. |
| Concurrencia | En programación significa cuando tareas se ejecutan de manera simultanea. |
| resume() | Nombre o etiqueta que clasifica a los usuarios con un conjunto de características comunes que generalmente conduce a un conjunto de permisos comunes. Los grupos pueden definirse ampliamente y reflejar los grupos de la vida real como "admin" o "manager”. |
| join() | Al utilizar este método nos aseguramos que no se ejecute otro hilo hasta que éste finalice |
| sleep(long ms) | Pone el hilo en pausa el tiempo en milisegundos que le introduzcamos |
| yield() | Pausa el hilo en ejecución para permitir la ejecución de otros |
| start() | Indica al intérprete de Java que cree un contexto del hilo del sistema y comience a ejecutarlo |
| IDE | Entorno de desarrollo integrado (IDE), es un software que permite desarrollar aplicaciones con herramientas que ayudan al programador a organizar su flujo de trabajo y agilizando todo el proceso de diseño de software, a través de una interfaz gráfica. |
| stop() | Provoca que el hilo se detenga de manera inmediata |
| resume() | Reanuda un hilo suspendido |
| setPriority() | Asigna la prioridad al hilo indicada por el valor pasado como parámetro |
| getPriority() | Devuelve la prioridad del hilo de ejecución en curso |
| setName() | Asigna un nombre para identificar a los hilos de una forma más cómoda |
| getName() | Devuelve el nombre asignado |

## PROTOCOLOS TCP/UDP.

TCP y UDP son dos protocolos de transmisión de datos diferentes [6], pero complementarios entre sí. Toda vez que el protocolo IP ha determinado la manera en que serán transmitidos los datos de un mensaje entre un dispositivo y otro, los protocolos TCP y UDP entran en acción para codificarlo y enviarlo a la dirección determinada por el protocolo IP.

El protocolo TCP [7] es un protocolo de la capa de transporte que es orientado a conexión, esto significa que antes de intercambiar los datos reales hay un paso previo para establecer una comunicación. Este protocolo también garantiza que toda la transmisión de los datos se hace sin errores, el propio TCP se encarga de reenviar los datos nuevamente en caso de que el receptor no los reciba a tiempo o los reciba dañados, además, garantiza también el orden, por lo que nos aseguramos de que los procesos van a recibir todos los datos en orden desde su origen. Establece una conexión entre el dispositivo emisor y el dispositivo receptor y verifica de forma continua la emisión y recepción de la información entre ambos.

Texto

Descripción generada automáticamente

Figura 2. Funcionamiento TCP

El protocolo UDP, no es orientado a conexión, no hay un paso previo en la comunicación, sino que se envían los datos directamente. Este protocolo no garantiza que la transmisión se realice sin errores, aunque hará todo lo posible para que sí lo haga, además, tampoco garantiza el orden de los datagramas que el origen envíe al destino. La parte positiva de UDP es que tiene una cabecera muy pequeña y es muy rápido, ya que no hay una fase de establecimiento de la conexión Por esto, el protocolo UDP es el más utilizado por los servicios de transmisión de voz o vídeo en streaming, donde la velocidad de la transmisión es más importante que una posible pérdida de datos puntual.

Imagen que contiene Icono

Descripción generada automáticamente

Figura 3. Funcionamiento UDP

## SOCKETS.

Los sockets [8] son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor) Java por medio de la librería java.net nos provee dos clases: Socket para implementar la conexión desde el lado del cliente y ServerSocket que nos permitirá manipular la conexión desde el lado del servidor. Tanto el cliente como el servidor no necesariamente deben estar implementados en Java, solo deben conocer sus direcciones IP y el puerto por el cual se comunicarán.

Diagrama

Descripción generada automáticamente

Figura 4. Funcionamiento de una conexión Socket

El servidor estará a la espera de que un cliente se conecte a él usando el socket en el puerto del protocolo TCP/UDP que es el que garantiza la entrega de paquetes de datos en la misma orden, en que fueron mandados, recibirá los mensajes, los mostrará y cerrará la conexión. El Cliente establecerá la conexión con el servidor usando un socket en localhost y el puerto del protocolo TCP/UDP.

# PARTE PRÁCTICA

## SOCKET SERVIDOR

### 2.1.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.

Imagen que contiene Forma

Descripción generada automáticamente

Figura 5. Logotipo identificativo de Netbeans

Primero dentro del disco C y cree una carpeta llamada DISTRIBUIDAS, dentro de esta crea otra subcarpeta con el nombre de Aplicativo-Socket a su vez dentro de esta última cree 2 carpetas, una con el nombre de “aplicativo” y otra con el nombre de “documentación”. Una vez haya creado las carpetas abre Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java with Ant -> Java Application

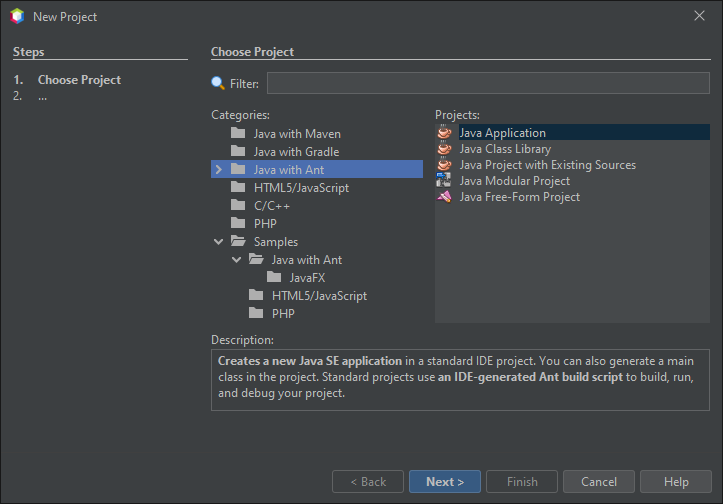


Figura 6. Selección del tipo de aplicación a desarrollar

En este caso para la aplicación de chat se crearán dos proyectos, uno para el servidor y otro para el cliente. A continuación, se detallará la creación de la parte del servidor. Le da el siguiente nombre SocketsChat\_Servidor\_Grupo6, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad. En este caso para la aplicación de chat se crearán dos proyectos, uno para el servidor y otro para el cliente. A continuación se detallará la creación de la parte del servidor.

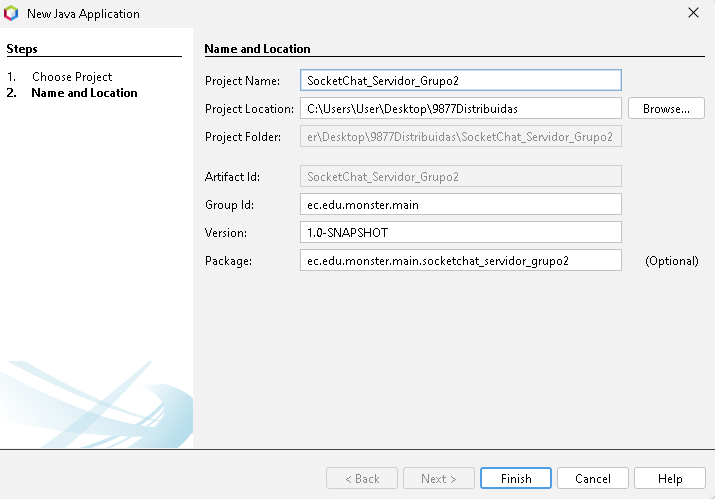


Figura 7. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Figura 8. Estructura del proyecto generado

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representará un elemento de la arquitectura, es decir que existirá un paquete visto, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 9. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

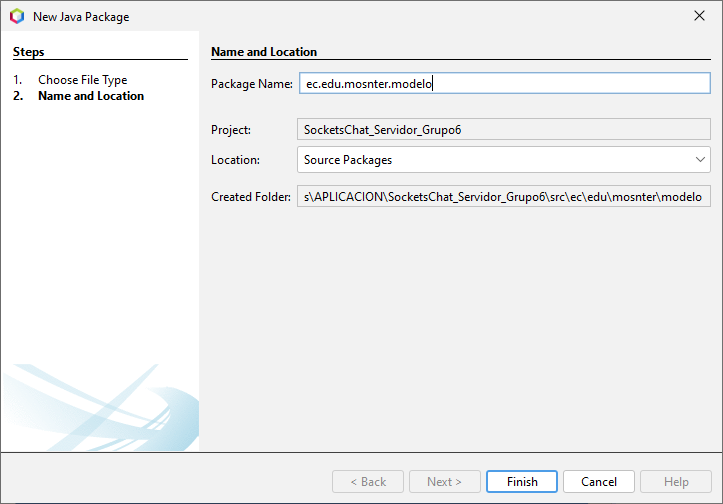


Figura 10. Selección del nombre de los paquetes

Repita el proceso hasta tener los 3 paquetes creados y cree un paquete llamado imagenes.

Una vez se tenga creado los 4 paquetes se puede eliminar el paquete ec.edu.monster.main junto con el archivo que contiene pues se definirá una clase de la vista para correr el proyecto.

Interfaz de usuario gráfica, Aplicación, Tabla, Word

Descripción generada automáticamente

Figura 11. Eliminación del paquete main.

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente JPanel Form pues se creará una interfaz gráfica tanto para el servidor como para el cliente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 12. Creación de clase Java.

Le da el nombre de “FormularioPrincipal”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista. Esta clase será la principal para correr el proyecto, en esta se visualizarán las acciones de parte del cliente, el estado del servidor y las conexiones, los errores y las operaciones que se realicen.

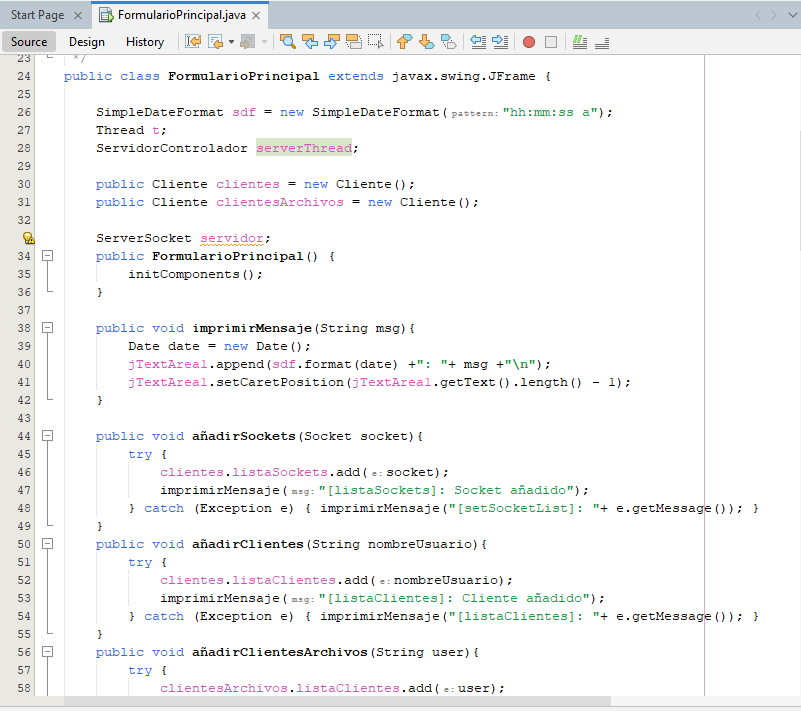


Figura 13. Codificación archivo FormularioPrincipal.java

Se incluirá dentro de paquete “imagenes” la siguiente imagen llamada servidor\_2.



Figura 14. Imagen del servidor

En la parte del diseño se colocan los elementos de la siguiente manera.

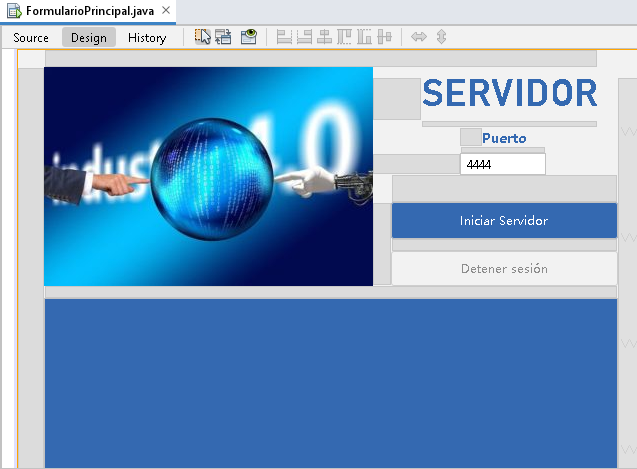


Figura 15. Diseño del FormularioPrincipal.java

.

Tabla 2. Código de FormularioPrincipal.java

package ec.edu.monster.vista;

import ec.edu.monster.controlador.ServidorControlador;

import ec.edu.monster.controlador.ClientesControlador;

import ec.edu.monster.modelo.Cliente;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Vector;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author USER

\*/

public class FormularioPrincipal extends javax.swing.JFrame {

SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss a");

Thread t;

ServidorControlador serverThread;

public Cliente clientes = new Cliente();

public Cliente clientesArchivos = new Cliente();

ServerSocket servidor;

public FormularioPrincipal() {

initComponents();

}

public void imprimirMensaje(String msg){

Date date = new Date();

jTextArea1.append(sdf.format(date) +": "+ msg +"\n");

jTextArea1.setCaretPosition(jTextArea1.getText().length() - 1);

}

public void añadirSockets(Socket socket){

try {

clientes.listaSockets.add(socket);

imprimirMensaje("[listaSockets]: Socket añadido");

} catch (Exception e) { imprimirMensaje("[setSocketList]: "+ e.getMessage()); }

}

public void añadirClientes(String nombreUsuario){

try {

clientes.listaClientes.add(nombreUsuario);

imprimirMensaje("[listaClientes]: Cliente añadido");

} catch (Exception e) { imprimirMensaje("[listaClientes]: "+ e.getMessage()); }

}

public void añadirClientesArchivos(String user){

try {

clientesArchivos.listaClientes.add(user);

} catch (Exception e) { }

}

public void añadirSocketsClientesArchivos(Socket soc){

try {

clientesArchivos.listaSockets.add(soc);

} catch (Exception e) { }

}

public Socket obtenerClientes(String cliente){

Socket tsoc = null;

for(int x=0; x < clientes.listaClientes.size(); x++){

if(clientes.listaClientes.get(x).equals(cliente)){

tsoc = (Socket) clientes.listaSockets.get(x);

break;

}

}

return tsoc;

}

public void quitarClienteLista(String cliente){

try {

for(int x=0; x < clientes.listaClientes.size(); x++){

if(clientes.listaClientes.elementAt(x).equals(cliente)){

clientes.listaClientes.removeElementAt(x);

clientes.listaSockets.removeElementAt(x);

imprimirMensaje("[Salió]: "+ cliente);

break;

}

}

} catch (Exception e) {

imprimirMensaje("[RemovedException]: "+ e.getMessage());

}

}

public Socket obtenerSocketsClientesArchivos(String nombreUsuario){

Socket tsoc = null;

for(int x=0; x < clientesArchivos.listaClientes.size(); x++){

if(clientesArchivos.listaClientes.elementAt(x).equals(nombreUsuario)){

tsoc = (Socket) clientesArchivos.listaSockets.elementAt(x);

break;

}

}

return tsoc;

}

public void quitarClienteArchivo(String nombreUsuario){

for(int x=0; x < clientesArchivos.listaClientes.size(); x++){

if(clientesArchivos.listaClientes.elementAt(x).equals(nombreUsuario)){

try {

Socket rSock = obtenerSocketsClientesArchivos(nombreUsuario);

if(rSock != null){

rSock.close();

}

clientesArchivos.listaClientes.removeElementAt(x);

clientesArchivos.listaSockets.removeElementAt(x);

imprimirMensaje("[FileSharing]: Cancelar "+ nombreUsuario);

} catch (IOException e) {

imprimirMensaje("[FileSharing]: "+ e.getMessage());

imprimirMensaje("[FileSharing]: No se puede cancelar "+ nombreUsuario);

}

break;

}

}

}

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

// TODO add your handling code here:

int port = Integer.parseInt(jTextField1.getText());

serverThread = new ServidorControlador(port, this);

t = new Thread(serverThread);

t.start();

new Thread(new ClientesControlador(this)).start();

jButton1.setEnabled(false);

jButton2.setEnabled(true);

}

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

// TODO add your handling code here:

int confirm = JOptionPane.showConfirmDialog(null, "¿Realmente desea desconectar el Servidor?");

if(confirm == 0){

serverThread.stop();

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

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

public void run() {

new FormularioPrincipal().setVisible(true);

}

});

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

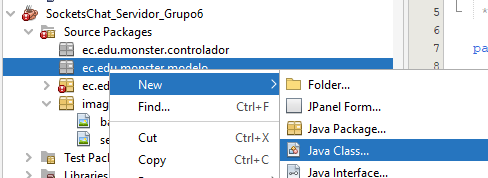


Figura 16. Creación de clase Java.

Le da el nombre de “Cliente”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de modelo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 17. Codificación archivo Cliente.java.

Esta clase permite tener un modelo que almacene una lista de los nombres y el socket de los clientes que entren en el servidor.

package ec.edu.monster.modelo;

import java.util.Vector;

public class Cliente {

public Vector listaSockets = new Vector();

public Vector listaClientes = new Vector();

public Cliente() {

}

public Vector getListaSockets() {

return listaSockets;

}

public void setListaSockets(Vector listaSockets) {

this.listaSockets = listaSockets;

}

public Vector getListaClientes() {

return listaClientes;

}

public void setListaClientes(Vector listaClientes) {

this.listaClientes = listaClientes;

}

}

Tabla 3. Codificación del archivo Cliente.java

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 18. Creación de clase Java.

Le da el nombre de “ClientesControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 19.Codificación archivo ClientesControlador.

Esta clase permite manejar el hilo de la lista de clientes y sus sockets que llegan al servidor, imprimiéndolos en el proceso a la consola del servidor dentro de la interfaz del cliente.

Tabla 4. Codificación del archivo ClientesControlador

package ec.edu.monster.controlador;

import ec.edu.monster.vista.FormularioPrincipal;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class ClientesControlador implements Runnable {

FormularioPrincipal main;

public ClientesControlador(FormularioPrincipal main){

this.main = main;

}

@Override

public void run() {

try {

while(!Thread.interrupted()){

String msg = "";

for(int x=0; x < main.clientes.listaClientes.size(); x++){

msg = msg+" "+ main.clientes.listaClientes.elementAt(x);

}

for(int x=0; x < main.clientes.listaSockets.size(); x++){

Socket tsoc = (Socket) main.clientes.listaSockets.elementAt(x);

DataOutputStream dos = new DataOutputStream(tsoc.getOutputStream());

/\*\* CMD\_ONLINE [user1] [user2] [user3] \*\*/

if(msg.length() > 0){

dos.writeUTF("CMD\_ONLINE "+ msg);

}

}

Thread.sleep(1900);

}

} catch(InterruptedException e){

main.imprimirMensaje("[InterruptedException]: "+ e.getMessage());

} catch (IOException e) {

main.imprimirMensaje("[IOException]: "+ e.getMessage());

}

}

}

Cree una nueva clase en el paquete del controlador.

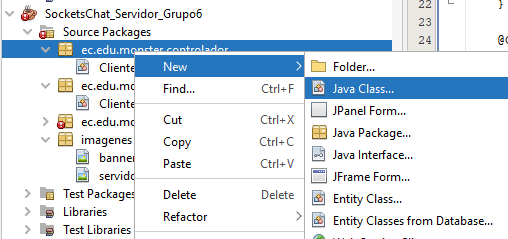


Figura 20. Creación de la clase

Le da el nombre de “SocketsControlador”, y proceda a colocar el siguiente código para construir la clase en el paquete de controlador.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Figura 21. Codificación archivo SocketsControlador

Esta clase es la más importante de la parte del servidor pues controla los sockets del proyecto y gestiona todas las acciones y los hilos del servidor que van a interactuar directamente con el cliente, como lo es la conexión, el chat de mensajes, el envío y recepción de archivos y la visualización de la información en el servidor.

package ec.edu.monster.controlador;

import ec.edu.monster.vista.FormularioPrincipal;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

import java.util.StringTokenizer;

public class SocketsControlador implements Runnable {

Socket socket;

FormularioPrincipal main;

DataInputStream dis;

StringTokenizer st;

String cliente, nombreClienteArchivo;

private final int BUFFER\_SIZE = 100;

public SocketsControlador(Socket socket, FormularioPrincipal main) {

this.main = main;

this.socket = socket;

try {

dis = new DataInputStream(socket.getInputStream());

} catch (IOException e) {

main.imprimirMensaje("[SocketThreadIOException]: " + e.getMessage());

}

}

Tabla 5. Codificación del archivo SocketsControlador

private void crearConexion(String receiver, String sender, String filename) {

try {

main.imprimirMensaje("[Conexion creada]: Conexión creada entre los usuarios para la tranferencia de archivos");

Socket s = main.obtenerClientes(receiver);

if (s != null) { // Client đã tồn tại

main.imprimirMensaje("[Conexion creada]: Socket OK");

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

main.imprimirMensaje("[createConnection]: DataOutputStream OK");

String format = "CMD\_FILE\_XD " + sender + " " + receiver + " " + filename;

dosS.writeUTF(format);

main.imprimirMensaje("[Conexion creada]: " + format);

} else {

main.imprimirMensaje("[Conexion creada]: El cliente solicitado no fue encontrado '" + receiver + "'");

DataOutputStream dos = new DataOutputStream(socket.getOutputStream());

dos.writeUTF("CMD\_SENDFILEERROR " + "El destinatario '" + receiver + "' no se encuentra conectado");

}

} catch (IOException e) {

main.imprimirMensaje("[createConnection]: " + e.getLocalizedMessage());

}

}

@Override

public void run() {

try {

while (true) {

String data = dis.readUTF();

st = new StringTokenizer(data);

String CMD = st.nextToken();

switch (CMD) {

case "CMD\_JOIN":

String nombreCliente = st.nextToken();

cliente = nombreCliente;

main.añadirClientes(nombreCliente);

main.añadirSockets(socket);

main.imprimirMensaje("[Client]: " + nombreCliente + " Se ha unido al chat!");

break;

case "CMD\_CHAT":

String desde = st.nextToken();

String para = st.nextToken();

String msg = "";

while (st.hasMoreTokens()) {

msg = msg + " " + st.nextToken();

}

Socket tsoc = main.obtenerClientes(para);

try {

DataOutputStream dos = new DataOutputStream(tsoc.getOutputStream());

String content = desde + ": " + msg;

dos.writeUTF("CMD\_MESSAGE " + content);

main.imprimirMensaje("[Message]:De " + desde + " para " + para + " : " + msg);

} catch (IOException e) {

main.imprimirMensaje("[IOException]: No se puede enviar el mensaje a " + para);

}

break;

} catch (IOException e) {

main.imprimirMensaje("[IOException]: No se puede enviar el mensaje a " + para);

}

break;

case "CMD\_CHATALL":

String chatall\_from = st.nextToken();

String chatall\_msg = "";

while (st.hasMoreTokens()) {

chatall\_msg = chatall\_msg + " " + st.nextToken();

}

String chatall\_content = chatall\_from + " " + chatall\_msg;

for (int x = 0; x < main.clientes.listaClientes.size(); x++) {

if (!main.clientes.listaClientes.elementAt(x).equals(chatall\_from)) {

try {

Socket tsoc2 = (Socket) main.clientes.listaSockets.elementAt(x);

DataOutputStream dos2 = new DataOutputStream(tsoc2.getOutputStream());

dos2.writeUTF("CMD\_MESSAGE " + chatall\_content);

} catch (IOException e) {

main.imprimirMensaje("[CMD\_CHATALL]: " + e.getMessage());

}

}

}

main.imprimirMensaje("[CMD\_CHATALL]: " + chatall\_content);

break;

case "CMD\_SHARINGSOCKET":

main.imprimirMensaje("CMD\_SHARINGSOCKET : Estableciendo conexión para el intercambio de archivos...");

String file\_sharing\_username = st.nextToken();

nombreClienteArchivo = file\_sharing\_username;

main.añadirClientesArchivos(file\_sharing\_username);

main.añadirSocketsClientesArchivos(socket);

main.imprimirMensaje("CMD\_SHARINGSOCKET : Usuario: " + file\_sharing\_username);

main.imprimirMensaje("CMD\_SHARINGSOCKET : Aceptar la tranferencia de archivos");

break;

case "CMD\_SENDFILE":

main.imprimirMensaje("CMD\_SENDFILE : El cliente está enviando un archivo...");

/\*

Format: CMD\_SENDFILE [Filename] [Size] [Recipient] [Consignee] desde: Sender Format

Format: CMD\_SENDFILE [Filename] [Size] [Consignee] to Receiver Format

\*/

String file\_name = st.nextToken();

String filesize = st.nextToken();

String sendto = st.nextToken();

String consignee = st.nextToken();

main.imprimirMensaje("CMD\_SENDFILE : De: " + consignee);

main.imprimirMensaje("CMD\_SENDFILE : Para: " + sendto);

/\*\*

\* Obtener el socket del cliente

\*/

main.imprimirMensaje("CMD\_SENDFILE : Servidor listo para conexiones..");

Socket cSock = main.obtenerSocketsClientesArchivos(sendto); /\* Consignee Socket \*/

/\* Verificar si existe conexion con el destinatario \*/

if (cSock != null) { /\* Exists \*/

try {

main.imprimirMensaje("CMD\_SENDFILE : Se ha establecido la conexión");

main.imprimirMensaje("CMD\_SENDFILE : Enviando archivo...");

DataOutputStream cDos = new DataOutputStream(cSock.getOutputStream());

cDos.writeUTF("CMD\_SENDFILE " + file\_name + " " + filesize + " " + consignee);

InputStream input = socket.getInputStream();

OutputStream sendFile = cSock.getOutputStream();

byte[] buffer = new byte[BUFFER\_SIZE];

int cnt;

while ((cnt = input.read(buffer)) > 0) {

sendFile.write(buffer, 0, cnt);

}

sendFile.flush();

sendFile.close();

main.quitarClienteArchivo(sendto);

main.quitarClienteArchivo(consignee);

main.imprimirMensaje("CMD\_SENDFILE : El archivo fue enviado exitosamente");

} catch (IOException e) {

main.imprimirMensaje("[CMD\_SENDFILE]: " + e.getMessage());

}

} else {

main.quitarClienteArchivo(consignee);

main.imprimirMensaje("CMD\_SENDFILE : Client '" + sendto + "Extraviado");

DataOutputStream dos = new DataOutputStream(socket.getOutputStream());

dos.writeUTF("CMD\_SENDFILEERROR " + "Client '" + sendto + "' no se encuentra, se cerrará el uso compartido de archivos.");

}

break;

case "CMD\_SEND\_FILE\_ERROR":

String eReceiver = st.nextToken();

String eMsg = "";

while (st.hasMoreTokens()) {

eMsg = eMsg + " " + st.nextToken();

}

try {

Socket eSock = main.obtenerSocketsClientesArchivos(eReceiver);

DataOutputStream eDos = new DataOutputStream(eSock.getOutputStream());

eDos.writeUTF("CMD\_RECEIVE\_FILE\_ERROR " + eMsg);

} catch (IOException e) {

main.imprimirMensaje("[CMD\_RECEIVE\_FILE\_ERROR]: " + e.getMessage());

}

break;

case "CMD\_SEND\_FILE\_ACCEPT":

String aReceiver = st.nextToken();

String aMsg = "";

while (st.hasMoreTokens()) {

aMsg = aMsg + " " + st.nextToken();

}

try {

Socket aSock = main.obtenerSocketsClientesArchivos(aReceiver);

DataOutputStream aDos = new DataOutputStream(aSock.getOutputStream());

aDos.writeUTF("CMD\_RECEIVE\_FILE\_ACCEPT " + aMsg);

} catch (IOException e) {

main.imprimirMensaje("[CMD\_RECEIVE\_FILE\_ERROR]: " + e.getMessage());

}

break;

default:

main.imprimirMensaje("[CMDException]: El comando ingresado para la tranferencia es desconocido " + CMD);

break;

}

}

} catch (IOException e) {

System.out.println(cliente);

System.out.println("Archivo compartido: " + nombreClienteArchivo);

main.quitarClienteLista(cliente);

if (nombreClienteArchivo != null) {

main.quitarClienteArchivo(nombreClienteArchivo);

}

main.imprimirMensaje("[SocketThread]: La conexión creada con el cliente esta cerrada");

}

}

}

Cree otra clase en el paquete del controlador.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 22. Creación de la clase

Le da el nombre de “ServidorControlador”, y proceda a colocar el siguiente código para construir la clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 23. Codificación archivo ServidorControlador

Esta clase se encarga de controlar los hilos de la ejecución y detenimiento del servidor.

package ec.edu.monster.controlador;

import ec.edu.monster.vista.FormularioPrincipal;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class ServidorControlador implements Runnable {

ServerSocket server;

FormularioPrincipal main;

boolean keepGoing = true;

public ServidorControlador(int port, FormularioPrincipal main){

main.imprimirMensaje("[Servidor]: Corriendo en el puerto " + port + ". Cargando... ");

try {

this.main = main;

server = new ServerSocket(port);

main.imprimirMensaje("[Servidor]: Servidor iniciado exitosamente");

}

catch (IOException e) { main.imprimirMensaje("[IOException]: "+ e.getMessage()); }

}

Tabla 6. Codificación del archivo ServidorControlador

@Override

public void run() {

try {

while(keepGoing){

Socket socket = server.accept();

new Thread(new SocketsControlador(socket, main)).start();

}

} catch (IOException e) {

main.imprimirMensaje("[ServerThreadIOException]: "+ e.getMessage());

}

}

public void stop(){

try {

server.close();

keepGoing = false;

System.out.println("Servidor apagado, se han cortado todas las conexiones");

System.exit(0);

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

### EJECUCIÓN DEL PROYECTO.

Para ejecutar el proyecto se da clic derecho en el nombre de este y luego en “Run”, después se selecciona que la clase main sea “FormularioPrincipal” y listo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 24. Ejecución del proyecto

Al ejecutar el proyecto se puede observar una ventana similar a la siguiente donde se encuentran los botones para iniciar o detener el servidor, junto con un JTextField para ingresar el número del puerto.

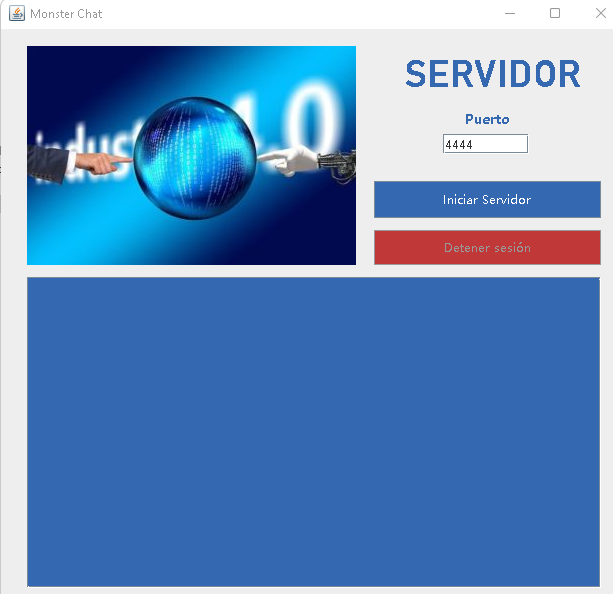


Figura 25. Resultado de la ejecución

Se puede dar clic en el botón “Iniciar Servidor” para iniciarlo y en la parte inferior se visualizará el estado del mismo, así como el puerto en el que está corriendo.

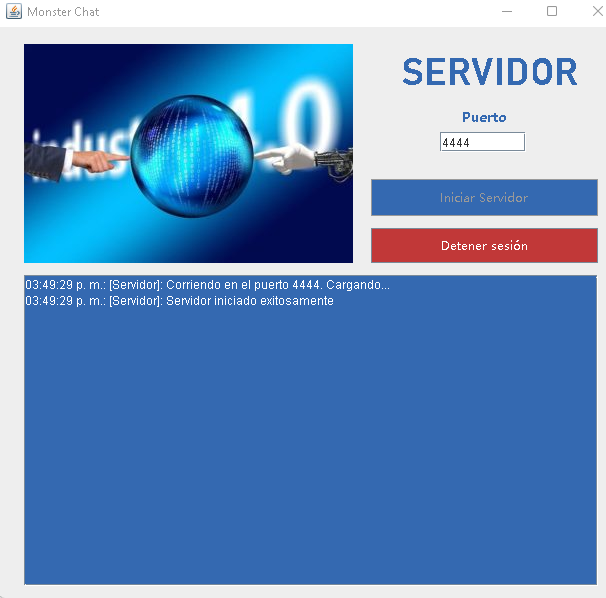


Figura 26. Inicio del servidor

## SOCKET CLIENTE

### 2.1.1 CREACIÓN DEL PROYECTO.

Abra el IDE de desarrollo Apache Netbeans 14.



Figura 27. Logotipo identificativo de Netbeans

Primero dentro del disco C y cree una carpeta llamada DISTRIBUIDAS, dentro de esta crea otra subcarpeta con el nombre de Aplicativo-Socket a su vez dentro de esta última cree 2 carpetas, una con el nombre de “aplicativo” y otra con el nombre de “documentación”. Una vez haya creado las carpetas abre Apache Netbeans y se sitúa en la Pestaña superior y selecciona File->New Project -> Java with Ant -> Java Application

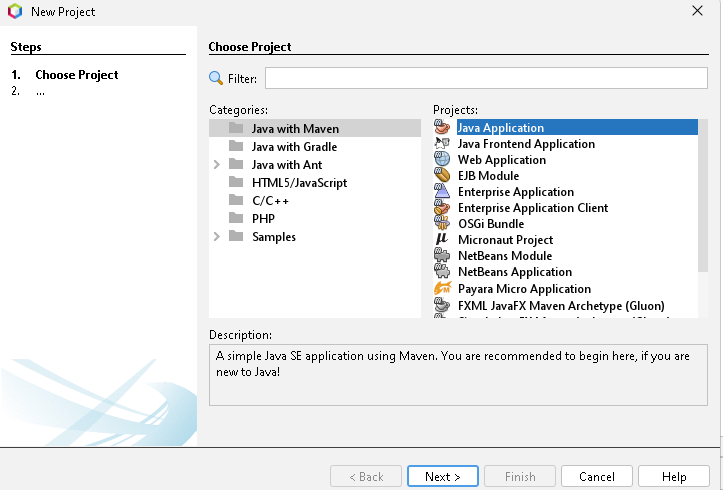


Figura 28. Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre SocketsChat\_Cliente\_Grupo6, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad

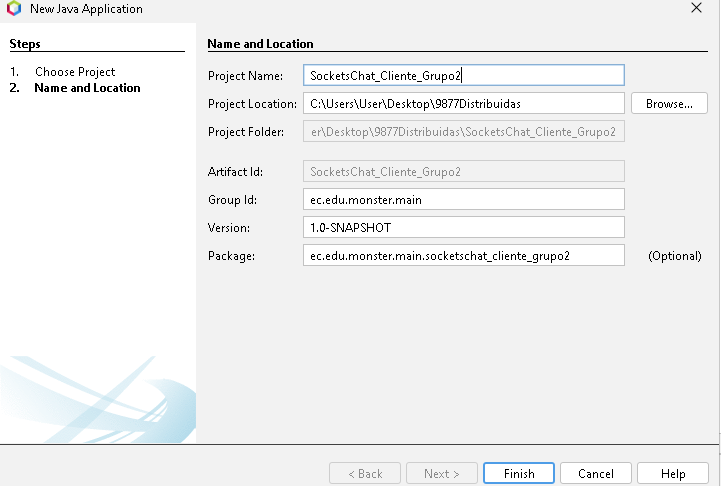


Figura 29. Nombre del proyecto

De clic en finalizar y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

Texto

Descripción generada automáticamente

Figura 30. Estructura del proyecto generado.

### CREACIÓN DE CARPETAS PARA IMÁGENES Y AUDIO

Dentro del proyecto en la ruta Aplicativo-Sockets -> SocketsChat\_Cliente -> src, se crea la carpeta llamada imágenes la cual contendrá las ilustraciones que se mostraran en la ejecución de la aplicación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 31. Carpeta de imágenes

De igual manera en la misma ruta se creará la carpeta llamada audio que guardará los sonidos que se emitirán al interactuar con la aplicación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 32. Carpeta de sonido

### Creación de paquetes MVC.

Para aplicar la arquitectura MVC es necesario crear 3 paquetes distintos, cada paquete representara a un elemento de la arquitectura, es decir que existirá un paquete vista, uno controlador y uno de modelo, para ello siga los siguientes pasos.

De clic en sources packages->new->java package

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 33. Creación de paquete.

En el nombre del paquete coloque ec.edu.monster.(modelo-vista-controlador) según requiera.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 34. Selección del nombre de los paquetes

Repita el proceso hasta tener los 3 paquetes creados

Una vez se tenga creado los 4 paquetes se puede eliminar el paquete ec.edu.monster.main junto con el archivo que contiene pues se definirá una clase de la vista para correr el proyecto.

Interfaz de usuario gráfica, Aplicación, Tabla, Word

Descripción generada automáticamente

Figura 35. Eliminación del paquete main

### CODIFICACIÓN DE LOS ARCHIVOS DE LA VISTA

De clic derecho en el paquete vista, luego nuevo y finalmente JPanel Form.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 36. Creación de clase Java

Le da el nombre de “PantallaPrincipal”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de vista.

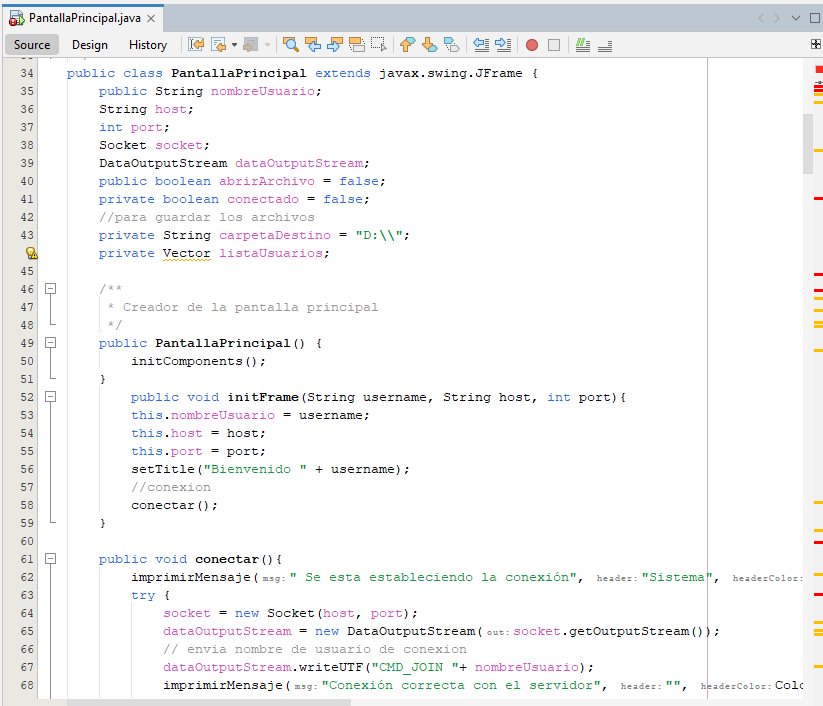


Figura 37. Codificación archivo PantallaPrincipal

En la parte del diseño se colocan los elementos de la siguiente manera.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 38. Diseño del formulario de PantallaPrincipal.java

Tabla 7. Código del archivo PantallaPrincipal.

public class PantallaPrincipal extends javax.swing.JFrame {

public String nombreUsuario;

String host;

int port;

Socket socket;

DataOutputStream dataOutputStream;

public boolean abrirArchivo = false;

private boolean conectado = false;

//para guardar los archivos

private String carpetaDestino = "D:\\";

private Vector listaUsuarios;

/\*\*

\* Creador de la pantalla principal

\*/

public PantallaPrincipal() {

initComponents();

}

public void initFrame(String username, String host, int port){

this.nombreUsuario = username;

this.host = host;

this.port = port;

setTitle("Bienvenido " + username);

//conexion

conectar();

}

public void conectar(){

imprimirMensaje(" Se esta estableciendo la conexión", "Sistema", Color.BLACK, new Color(147,147,147), "", true, true, "", false, true);

try {

socket = new Socket(host, port);

dataOutputStream = new DataOutputStream(socket.getOutputStream());

// envia nombre de usuario de conexion

dataOutputStream.writeUTF("CMD\_JOIN "+ nombreUsuario);

imprimirMensaje("Conexión correcta con el servidor", "", Color.WHITE, new Color(115,173,115), "", true, true, "", false, true);

imprimirMensaje("Puedes comenzar a chatear", "", Color.WHITE, new Color(147,147,147), "", true, true, "", false, true);

// inicia hilo de cliente

new Thread(new MensajesControlador(socket, this)).start();

btnEnviar.setEnabled(true);

// se ha conectado

conectado = true;

}

catch(IOException e) {

conectado = false;

JOptionPane.showMessageDialog(this, "Ocurrio un error al conectar intentelo mas tarde","Conexión fallida",JOptionPane.ERROR\_MESSAGE);

imprimirMensaje("[IOException]: "+ e.getMessage(), "Error", Color.RED, Color.RED, "", true, true, "", false, false);

}

}

public boolean verificarConexion(){

return this.conectado;

}

public void imprimirMensaje(String msg, String header, Color headerColor, Color contentColor, String fontFamilyHeader, boolean boldHeader, boolean italicHeader, String fontFamilyContent, boolean boldContent, boolean italicContent){

txpPanelChat.setEditable(true);

imprimirEncabezado(header, headerColor, fontFamilyHeader, boldHeader, italicHeader);

imprimirContenido(msg, contentColor, fontFamilyContent, boldContent, italicContent);

txpPanelChat.setEditable(false);

}

public void imprimirMensajePropio(String msg, String header){

txpPanelChat.setEditable(true);

imprimirEncabezado(header, new Color(52,105,177), "", true, false);

imprimirContenido(msg, new Color(72,149,209), "", false, false);

txpPanelChat.setEditable(false);

}

public void imprimirEncabezado(String header, Color color, String fontFamily, boolean bold, boolean italic){

if (!header.isEmpty()) {

int len = txpPanelChat.getDocument().getLength();

txpPanelChat.setCaretPosition(len);

txpPanelChat.setCharacterAttributes(Estilos.styleMessageContent(color, fontFamily.isEmpty() ? "Malgun Gothic" : fontFamily, 13, bold, italic), false);

txpPanelChat.replaceSelection(header+"\n");

}

}

public void imprimirContenido(String msg, Color color, String fontFamily, boolean bold, boolean italic){

int len = txpPanelChat.getDocument().getLength();

txpPanelChat.setCaretPosition(len);

txpPanelChat.setCharacterAttributes(Estilos.styleMessageContent(color, fontFamily.isEmpty() ? "Microsoft YaHei UI" : fontFamily, 13, bold, italic), false);

txpPanelChat.replaceSelection(msg +"\n\n");

}

public void añadirUsuarioConectados(Vector list){

imprimirListaUsuarios(list);

}

public void imprimirUsuariosConectados(Vector list){

try {

txpUsuariosConectados.setEditable(true);

txpUsuariosConectados.setContentType("text/html");

StringBuilder sb = new StringBuilder();

Iterator it = list.iterator();

sb.append("<html><table>");

while(it.hasNext()){

Object e = it.next();

URL url = obtenerImagen();

Icon icon = new ImageIcon(this.getClass().getResource("/images/online.png"));

sb.append("<tr><td><b>></b></td><td>").append(e).append("</td></tr>");

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

}

sb.append("</table></body></html>");

txpUsuariosConectados.removeAll();

txpUsuariosConectados.setText(sb.toString());

txpUsuariosConectados.setEditable(false);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

private void imprimirListaUsuarios(Vector list){

listaUsuarios = list;

txpUsuariosConectados.setEditable(true);

txpUsuariosConectados.removeAll();

txpUsuariosConectados.setText("");

Iterator i = list.iterator();

while(i.hasNext()){

Object e = i.next();

/\* nombre de usuario que envio el mensaje \*/

JPanel panel = new JPanel();

panel.setLayout(new FlowLayout(FlowLayout.LEFT));

panel.setBackground(Color.white);

Icon icon = new ImageIcon(this.getClass().getResource("/images/usuarioOnline\_25.png"));

JLabel label = new JLabel(icon);

label.setText(" "+ e);

panel.add(label);

int len = txpUsuariosConectados.getDocument().getLength();

txpUsuariosConectados.setCaretPosition(len);

txpUsuariosConectados.insertComponent(panel);

/\* nueva linea luego de cada mensaje \*/

saltarLinea();

}

txpUsuariosConectados.setEditable(false);

}

private void saltarLinea(){

int len = txpUsuariosConectados.getDocument().getLength();

txpUsuariosConectados.setCaretPosition(len);

txpUsuariosConectados.replaceSelection("\n");

}

public URL obtenerImagen(){

URL url = this.getClass().getResource("/images/online.png");

return url;

}

public void asignarTitulo(String s){

setTitle(s);

}

public String getCarpetaDestino(){

return this.carpetaDestino;

}

public String getHost(){

return this.host;

}

public int getPort(){

return this.port;

}

public String getNombreUsuario(){

return this.nombreUsuario;

}

public void actualizarArchivoAdjunto(boolean b){

this.abrirArchivo = b;

}

public void abrirCarpeta(){

JFileChooser chooser = new JFileChooser();

chooser.setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

int open = chooser.showDialog(this, "Carpeta abierta");

if(open == chooser.APPROVE\_OPTION){

carpetaDestino = chooser.getSelectedFile().toString()+"\\";

} else {

carpetaDestino = "D:\\";

}

}

private void btnCerrarSesionActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

int confirm = JOptionPane.showConfirmDialog(null, "¿Está seguro de querer salir de la sesión?");

if(confirm == 0){

try {

socket.close();

setVisible(false);

/\*\* Login Form \*\*/

new FormularioLogin().setVisible(true);

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

private void txtMensajeActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try {

String content = nombreUsuario+" "+ evt.getActionCommand();

dataOutputStream.writeUTF("CMD\_CHATALL "+ content);

imprimirMensajePropio(" "+evt.getActionCommand(), nombreUsuario);

txtMensaje.setText("");

} catch (IOException e) {

imprimirMensaje(" No se pudo enviar por falta de conexión!", "Error", Color.RED, Color.RED, "", true, true, "", false, true);

}

}

private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try {

String content = nombreUsuario+" "+ txtMensaje.getText();

dataOutputStream.writeUTF("CMD\_CHATALL "+ content);

imprimirMensajePropio(" "+txtMensaje.getText(), nombreUsuario);

txtMensaje.setText("");

} catch (IOException e) {

imprimirMensaje(" No se pudo enviar por falta de conexión!", "Error", Color.RED, Color.RED, "", true, true, "", false, true);

}

}

private void btnArchivoActionPerformed(java.awt.event.ActionEvent evt) {

if(!abrirArchivo){

FormularioEnviarArchivos s = new FormularioEnviarArchivos();

if(s.preparar(nombreUsuario, host, port, this, listaUsuarios)){

s.setLocationRelativeTo(null);

s.setVisible(true);

abrirArchivo = true;

} else {

JOptionPane.showMessageDialog(this, "No se pudo compartir el archivo!", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(PantallaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(PantallaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(PantallaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(PantallaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

/\* Create and display the form \*/

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

public void run() {

new PantallaPrincipal().setVisible(true);

}

});

}

Cree una nueva JPanel Form en el paquete de vista.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 39. Creación de clase Java.

Le da el nombre de “FormularioLogin”, y proceda a colocar el siguiente código para construir la clase en el paquete de vista.

Texto

Descripción generada automáticamente

Figura 40. Codificación archivo FormularioLogin

En la parte del diseño se colocan los elementos de la siguiente manera.



Figura 41. Diseño del formulario de FormularioLogin.java

Tabla 8. Código del archivo FormularioLogin

public class FormularioLogin extends javax.swing.JFrame {

public FormularioLogin() {

initComponents();

}

private void connectToServer(){

if(txtHost.getText().length() > 0 && txtPort.getText().length() > 0 && txtUsuario.getText().length() > 0){

if(txtUsuario.getText().length() <= 15){

/\* Limpiar nombre de usuario \*/

String username = txtUsuario.getText();

String u = username.replace(" ", "\_");

/\* Dirigir a la pantalla principal PantallaPrincipal \*/

PantallaPrincipal main = new PantallaPrincipal();

main.initFrame(u, txtHost.getText(), Integer.parseInt(txtPort.getText()));

// Verificar los usuarios que estan conectados

if(main.verificarConexion()){

main.setLocationRelativeTo(null);

main.setVisible(true);

setVisible(false);

}

} else {

JOptionPane.showMessageDialog(this, "Máximo 15 caractéres!", "Error", JOptionPane.ERROR\_MESSAGE);

}

} else {

JOptionPane.showMessageDialog(this, "Llene todos los campos!", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

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

public void run() {

new FormularioLogin().setVisible(true);

}

});

}

Cree una nueva JPanel Form en el paquete de vista.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 42. Creación de clase Java.

Le da el nombre de “FormularioEnviarArchivos”, y proceda a colocar el siguiente código para construir la clase en el paquete de vista.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 43. Codificación archivo FormularioEnviarArchivos

En la parte del diseño se colocan los elementos de la siguiente manera.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 44. Diseño del formulario de FormularioEnviarArchivos.java

public class FormularioEnviarArchivos extends javax.swing.JFrame {

private Socket socket;

private DataInputStream dis;

private DataOutputStream dos;

private String miNombreUsuario;

private String host;

private int port;

private StringTokenizer st;

private String destinatario;

private String archivo;

private PantallaPrincipal main;

private Object seleccionar;

// Crea form FormularioEnviarArchivos

public FormularioEnviarArchivos() {

initComponents();

pgbCarga.setVisible(false);

}

Tabla 9. Código del archivo FormularioEnviarArchivos

public boolean preparar(String u, String h, int p, PantallaPrincipal m, Vector lista){

this.host = h;

this.miNombreUsuario = u;

this.port = p;

this.main = m;

/\* COnexion al servidor \*/

try {

socket = new Socket(host, port);

dos = new DataOutputStream(socket.getOutputStream());

dis = new DataInputStream(socket.getInputStream());

// Format: CMD\_SHARINGSOCKET [sender]

String format = "CMD\_SHARINGSOCKET "+ miNombreUsuario;

dos.writeUTF(format);

System.out.println(format);

/\* hilo de envio \*/

new Thread(new EnviarArchivo(this, lista)).start();

return true;

} catch (IOException e) {

System.out.println(e.getMessage());

}

return false;

}

/\* enviar datos al servidor \*/

class EnviarArchivo implements Runnable{

private FormularioEnviarArchivos form;

private Vector listaUsuarios;

public EnviarArchivo(FormularioEnviarArchivos form, Vector listaUsuarios){

this.form = form;

this.listaUsuarios = listaUsuarios;

}

private void cerrarVentanaUsuario(){

try {

socket.close();

} catch (IOException e) {

System.out.println("[closeMe]: "+e.getMessage());

}

dispose();

}

@Override

public void run() {

try {

Iterator i = listaUsuarios.iterator();

int contador = 0;

form.limpiarListaUsuarios();

while(i.hasNext()){

Object e = i.next();

form.añadirListaUsuarios(" "+ e);

contador++;

}

if (contador == 0){

form.añadirListaUsuarios("No hay usuarios conectados");

}

while(!Thread.currentThread().isInterrupted()){

String data = dis.readUTF(); //

st = new StringTokenizer(data);

String cmd = st.nextToken(); //

switch(cmd){

case "CMD\_RECEIVE\_FILE\_ERROR": //

String msg = "";

while(st.hasMoreTokens()){

msg = msg+" "+st.nextToken();

}

form.updateAttachment(false);

JOptionPane.showMessageDialog(FormularioEnviarArchivos.this, msg, "Error", JOptionPane.ERROR\_MESSAGE);

this.cerrarVentanaUsuario();

break;

case "CMD\_RECEIVE\_FILE\_ACCEPT":

new Thread(new EnviarArchivosControlador(socket, archivo, destinatario, miNombreUsuario, FormularioEnviarArchivos.this, main)).start();

break;

case "CMD\_SENDFILEERROR":

String emsg = "";

while(st.hasMoreTokens()){

emsg = emsg +" "+ st.nextToken();

}

System.out.println(emsg);

JOptionPane.showMessageDialog(FormularioEnviarArchivos.this, emsg,"Error", JOptionPane.ERROR\_MESSAGE);

form.updateAttachment(false);

form.deshabilitarGUI(false);

form.updateBtn("Enviar archivo");

break;

case "CMD\_SENDFILERESPONSE":

/\*

Format: CMD\_SENDFILERESPONSE [username] [Message]

\*/

String rReceiver = st.nextToken();

String rMsg = "";

while(st.hasMoreTokens()){

rMsg = rMsg+" "+st.nextToken();

}

form.updateAttachment(false);

JOptionPane.showMessageDialog(FormularioEnviarArchivos.this, rMsg, "Error", JOptionPane.ERROR\_MESSAGE);

dispose();

break;

}

}

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

private void btnEnviarArchivoActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

if (cmbDestinatario.getItemCount() == 0){

destinatario = "";

} else {

destinatario = (String) cmbDestinatario.getSelectedItem();

}

archivo = txtRutaArchivo.getText();

if((destinatario.length() > 0) && (archivo.length() > 0)){

try {

// Format: CMD\_SEND\_FILE\_XD [sender] [receiver] [filename]

txtRutaArchivo.setText("");

String fname = getThisFilename(archivo);

String format = "CMD\_SEND\_FILE\_XD "+miNombreUsuario+" "+destinatario+" "+fname;

dos.writeUTF(format);

System.out.println(format);

updateBtn("Enviando...");

btnEnviarArchivo.setEnabled(false);

} catch (IOException e) {

System.out.println(e.getMessage());

}

}else{

JOptionPane.showMessageDialog(this, "Por favor llene los espacios","Error", JOptionPane.ERROR\_MESSAGE);

}

}

public void limpiarListaUsuarios(){

cmbDestinatario.removeAllItems();

}

public void añadirListaUsuarios(String usuario){

cmbDestinatario.addItem(usuario);

}

public void setMyTitle(String s){

setTitle(s);

}

public void closeThis(){

dispose();

}

public String getThisFilename(String path){

File p = new File(path);

String fname = p.getName();

return fname.replace(" ", "\_");

}

public void updateAttachment(boolean b){

main.actualizarArchivoAdjunto(b);

}

public void updateBtn(String str){

btnEnviarArchivo.setText(str);

}

public void updateProgress(int val){

pgbCarga.setValue(val);

}

public static void main(String args[]) {

/\*Inicio del programa\*/

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

public void run() {

new FormularioEnviarArchivos().setVisible(true);

}

});

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL MODELO

De clic derecho en el paquete modelo, luego nuevo y finalmente clase Java.

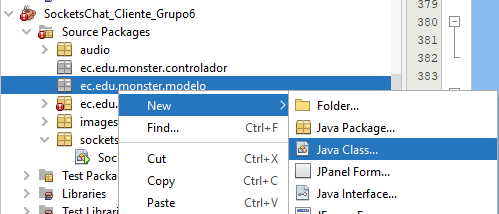


Figura 45. Creación de clase Java

Le da el nombre de “Estilos”, y proceda a colocar el siguiente código para construir su primera clase en el paquete modelo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 46. Codificación archivo Estilos

Tabla 10. Codificación del archivo Estilos

package ec.edu.monster.modelo;

import java.awt.Color;

import javax.swing.text.AttributeSet;

import javax.swing.text.SimpleAttributeSet;

import javax.swing.text.StyleConstants;

import javax.swing.text.StyleContext;

public class Estilos {

public static AttributeSet styleMessageContent(Color color, String fontFamily, int size, boolean bold, boolean italic){

StyleContext sc = StyleContext.getDefaultStyleContext();

AttributeSet aset = sc.addAttribute(SimpleAttributeSet.EMPTY, StyleConstants.Foreground, color);

aset = sc.addAttribute(aset, StyleConstants.FontFamily, fontFamily); // FontFamily

aset = sc.addAttribute(aset, StyleConstants.Alignment, StyleConstants.ALIGN\_JUSTIFIED);

aset = sc.addAttribute(aset, StyleConstants.FontSize, size);

aset = sc.addAttribute(aset, StyleConstants.Bold, bold);

aset = sc.addAttribute(aset, StyleConstants.Italic, italic);

return aset;

}

}

Cree una nueva clase en el paquete de modelo.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 47. Creación de clase Java.

Le da el nombre de “Sonidos”, y proceda a colocar el siguiente código para construir la clase en el paquete de modelo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 48. Codificación archivo Sonidos.java

Tabla 11. Codificación del archivo Sonidos.java

package ec.edu.monster.modelo;

public enum Sonidos {

MessageReceive("/audio/notificacion.wav", false), //

FileSharing("/audio/notificacionArchivo.wav", false); //

private Clip clip;

private boolean loop;

Sonidos(String filename, boolean loop){

try {

this.loop = loop;

URL url = this.getClass().getResource(filename);

AudioInputStream audioIS = AudioSystem.getAudioInputStream(url);

clip = AudioSystem.getClip();

clip.open(audioIS);

} catch (IOException | LineUnavailableException | UnsupportedAudioFileException e) {

System.out.println("[SoundEffect]" +e.getMessage());

}

}

public void reproducir(){

if(clip.isRunning()){

clip.stop(); // Stop Audio

}

// Reset Audio from the beginning

clip.setFramePosition(0);

clip.start();

// Check if audio reproducir contineously

if(loop){

clip.loop(Clip.LOOP\_CONTINUOUSLY);

}

}

public void detener(){

if(clip.isRunning()){

clip.stop(); // Stop Audio

}

}

}

### CODIFICACIÓN DE LOS ARCHIVOS DEL CONTROLADOR

De clic derecho en el paquete controlador, luego nuevo y finalmente clase Java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 49. Creación de clase Java.

Le da el nombre de “EnviarArchivosControlador”, y proceda a colocar el siguiente código para construir su primera clase en el paquete de controlador.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 50. Codificación archivo EnviarArchivosControlador

Esta clase tiene como función particular manejar el hilo que permite el envío de archivos de un usuario a otro que esté conectado en el chat.

Tabla 12. Codificación del archivo EnviarArchivosControlador

package ec.edu.monster.controlador;

public class EnviarArchivosControlador implements Runnable {

protected Socket socket;

private DataOutputStream dos;

protected FormularioEnviarArchivos form;

protected String file;

protected String receiver;

protected String sender;

PantallaPrincipal main;

protected DecimalFormat df = new DecimalFormat("##,#00");

private final int BUFFER\_SIZE = 100;

public EnviarArchivosControlador(Socket soc, String file, String receiver, String sender, FormularioEnviarArchivos frm, PantallaPrincipal main){

this.socket = soc;

this.file = file;

this.receiver = receiver;

this.sender = sender;

this.form = frm;

this.main = main;

}

@Override

public void run() {

try {

form.deshabilitarGUI(true);

System.out.println("Enviando archivo..!");

dos = new DataOutputStream(socket.getOutputStream());

File filename = new File(file);

int len = (int) filename.length();

int filesize = (int)Math.ceil(len / BUFFER\_SIZE);

String clean\_filename = filename.getName();

dos.writeUTF("CMD\_SENDFILE "+ clean\_filename.replace(" ", "\_") +" "+ filesize +" "+ receiver +" "+ sender);

System.out.println("Enviado por: "+ sender);

System.out.println("Destino: "+ receiver);

InputStream input = new FileInputStream(filename);

OutputStream output = socket.getOutputStream();

BufferedInputStream bis = new BufferedInputStream(input);

byte[] buffer = new byte[BUFFER\_SIZE];

int count, percent = 0;

while((count = bis.read(buffer)) > 0){

percent = percent + count;

int p = (percent / filesize);

form.updateProgress(p);

output.write(buffer, 0, count);

}

form.setMyTitle("Archivo enviado");

form.updateAttachment(false);

JOptionPane.showMessageDialog(form, "Archivo enviado exitosamente", "Éxito :D", JOptionPane.INFORMATION\_MESSAGE);

form.closeThis();

output.flush();

output.close();

System.out.println("Archivo enviado");

main.imprimirMensajePropio("Has enviado el archivo " + filename.getName() + " a " + receiver, sender);

} catch (IOException e) {

form.updateAttachment(false);

System.out.println("[SendFile]: "+ e.getMessage());

}

}

}

Cree una nueva clase en el paquete del controlador.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 51. Creación de clase Java.

Le da el nombre de “RecibirArchivosControlador”, y proceda a colocar el siguiente código para construir la clase en el paquete de controlador.

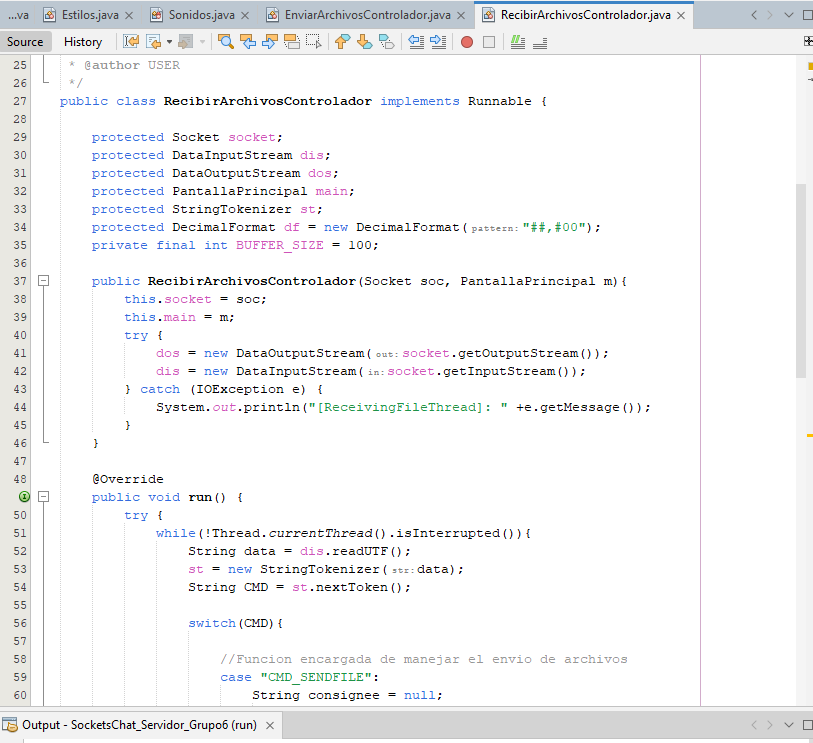


Figura 52. Codificación archivo RecibirArchivosControlador

Esta clase permite controlar el hilo de cuando se recibe un archivo, así como mostrar información en pantalla al cliente del estado del archivo enviado.

Tabla 13. Codificación del archivo RecibirArchivosControlador

package ec.edu.monster.controlador;

public class RecibirArchivosControlador implements Runnable {

protected Socket socket;

protected DataInputStream dis;

protected DataOutputStream dos;

protected PantallaPrincipal main;

protected StringTokenizer st;

protected DecimalFormat df = new DecimalFormat("##,#00");

private final int BUFFER\_SIZE = 100;

public RecibirArchivosControlador(Socket soc, PantallaPrincipal m){

this.socket = soc;

this.main = m;

try {

dos = new DataOutputStream(socket.getOutputStream());

dis = new DataInputStream(socket.getInputStream());

} catch (IOException e) {

System.out.println("[ReceivingFileThread]: " +e.getMessage());

}

}

@Override

public void run() {

try {

while(!Thread.currentThread().isInterrupted()){

String data = dis.readUTF();

st = new StringTokenizer(data);

String CMD = st.nextToken();

switch(CMD){

//Funcion encargada de manejar el envio de archivos

case "CMD\_SENDFILE":

String consignee = null;

try {

String filename = st.nextToken();

int filesize = Integer.parseInt(st.nextToken());

consignee = st.nextToken();

main.asignarTitulo("El archivo se está enviando");

System.out.println("El archivo se está enviando");

System.out.println("Desde: "+ consignee);

String path = main.getCarpetaDestino() + filename;

FileOutputStream fos = new FileOutputStream(path);

InputStream input = socket.getInputStream();

ProgressMonitorInputStream pmis = new ProgressMonitorInputStream(main, "Descargando, por favor espere ...", input);

BufferedInputStream bis = new BufferedInputStream(pmis);

byte[] buffer = new byte[BUFFER\_SIZE];

int count, percent = 0;

while((count = bis.read(buffer)) != -1){

percent = percent + count;

int p = (percent / filesize);

main.asignarTitulo("Descargando imágen "+ p +"%");

fos.write(buffer, 0, count);

}

fos.flush();

fos.close();

main.asignarTitulo("Usuario actual: " + main.getNombreUsuario());

JOptionPane.showMessageDialog(null, "El archivo fue guardado en \n'"+ path +"'");

System.out.println("El archivo se guardó existosamente en "+ path);

} catch (IOException e) {

DataOutputStream eDos = new DataOutputStream(socket.getOutputStream());

eDos.writeUTF("CMD\_SENDFILERESPONSE "+ consignee + " Se perdió la conexión, inténtalo de nuevo!");

System.out.println(e.getMessage());

main.asignarTitulo("Usuario actual: " + main.getNombreUsuario());

JOptionPane.showMessageDialog(main, e.getMessage(), "Exception", JOptionPane.ERROR\_MESSAGE);

socket.close();

}

break;

}

}

} catch (IOException e) {

System.out.println("[ReceivingFileThread]: " +e.getMessage());

}

}

Cree una nueva clase en el paquete del controlador.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 51. Creación de clase Java.

Le da el nombre de “MensajesControlador”, y proceda a colocar el siguiente código para construir la clase en el paquete de controlador.

Texto

Descripción generada automáticamente

Figura 52. Codificación archivo MensajesControlador

Esta clase se encarga de manejar los mensajes que se muestran en el chat, así como de desplegar la lista de usuarios conectados e iniciar con la transferencia de archivos.

package ec.edu.monster.controlador;

public class MensajesControlador implements Runnable{

Socket socket;

DataInputStream dis;

DataOutputStream dos;

PantallaPrincipal main;

StringTokenizer st;

protected DecimalFormat df = new DecimalFormat("##,#00");

public MensajesControlador(Socket socket, PantallaPrincipal main){

this.main = main;

this.socket = socket;

try {

dis = new DataInputStream(socket.getInputStream());

} catch (IOException e) {

main.imprimirMensaje("[IOException]: "+ e.getMessage(), "Error", Color.RED, Color.RED, "", true, true, "", false, true);

}

}

Tabla 14. Codificación del archivo MensajesControlador

@Override

public void run() {

try {

while(!Thread.currentThread().isInterrupted()){

String data = dis.readUTF();

st = new StringTokenizer(data);

String CMD = st.nextToken();

switch(CMD){

case "CMD\_MESSAGE":

Sonidos.MessageReceive.reproducir(); // Play Audio clip

String msg = "";

String frm = st.nextToken();

while(st.hasMoreTokens()){

msg = msg +" "+ st.nextToken();

}

main.imprimirMensaje(msg, frm, Color.BLACK, Color.BLACK, "", true, false, "", false, false);

break;

case "CMD\_ONLINE":

Vector online = new Vector();

while(st.hasMoreTokens()){

String list = st.nextToken();

if(!list.equalsIgnoreCase(main.nombreUsuario)){

online.add(list);

}

}

main.añadirUsuarioConectados(online);

break;

case "CMD\_FILE\_XD":

String sender = st.nextToken();

String receiver = st.nextToken();

String fname = st.nextToken();

int confirm = JOptionPane.showConfirmDialog(main, "De "+sender+"\nNombre archivo: "+fname+"\n¿Quiere aceptar el archivo?");

if(confirm == 0){

main.abrirCarpeta();

try {

dos = new DataOutputStream(socket.getOutputStream());

String format = "CMD\_SEND\_FILE\_ACCEPT "+sender+" Aceptado";

dos.writeUTF(format);

Socket fSoc = new Socket(main.getHost(), main.getPort());

DataOutputStream fdos = new DataOutputStream(fSoc.getOutputStream());

fdos.writeUTF("CMD\_SHARINGSOCKET "+ main.getNombreUsuario());

new Thread(new RecibirArchivosControlador(fSoc, main)).start();

} catch (IOException e) {

System.out.println("[CMD\_FILE\_XD]: "+e.getMessage());

}

} else {

try {

dos = new DataOutputStream(socket.getOutputStream());

String format = "CMD\_SEND\_FILE\_ERROR "+sender+"El usuario rechazó el envio o se desconecto!";

dos.writeUTF(format);

} catch (IOException e) {

System.out.println("[CMD\_FILE\_XD]: "+e.getMessage());

}

}

break;

default:

main.imprimirMensaje("[CMDException]: Comando inválido "+ CMD, "CMDException", Color.RED, Color.RED, "", true, true, "", false, true);

break;

}

}

} catch(IOException e){

main.imprimirMensaje(" Se ha perdido la conexión con el servidor, verifique el estado", "Error", Color.RED, Color.RED, "", true, true, "", false, true);

}

}

}

### EJECUCIÓN DEL PROYECTO.

Para ejecutar el proyecto del lado del cliente primero se debe ejecutar el proyecto correspondiente al Servidor e iniciarlo.

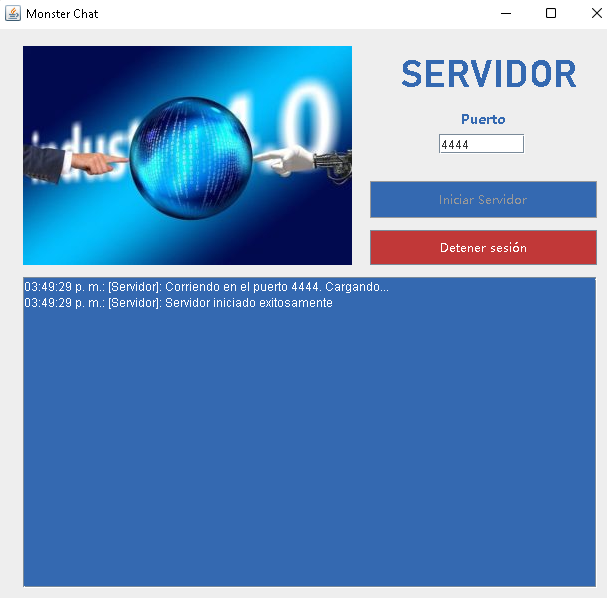


Figura 55. Ejecución del Servidor.

Con el servidor corriendo se puede ejecutar el proyecto correspondiente al cliente, primero se tendrá que ingresar el nombre de usuario, puerto e IP.

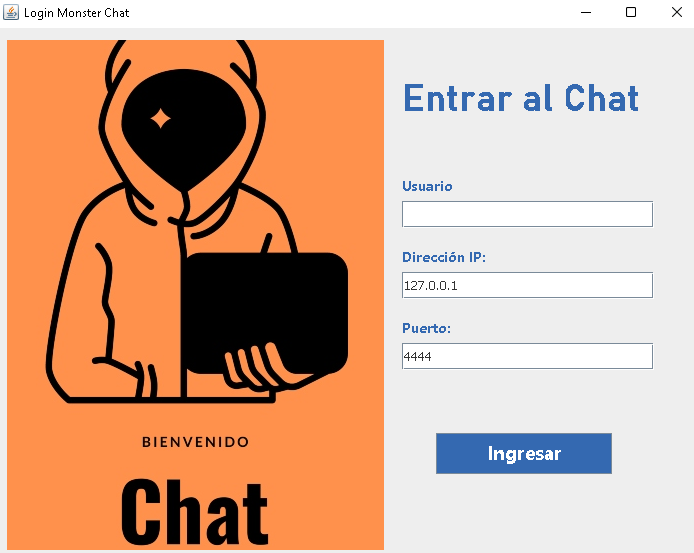


Figura 56. Ejecución del Cliente.

Al ingresar los datos se pasará al formulario del chat donde ya se podrá empezar a chatear y enviar archivos como imágenes, videos, pdf, words, etc.

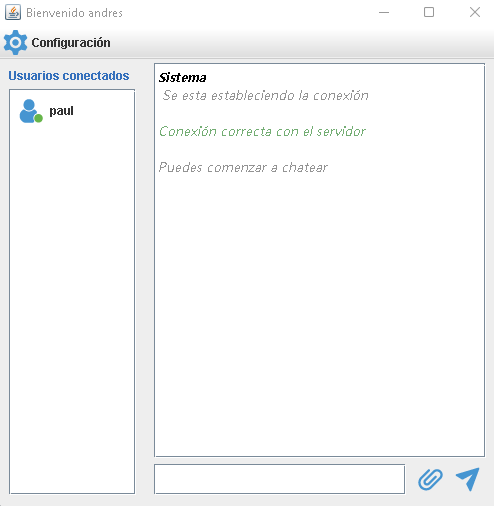


Figura 57. Ejecución del chat del cliente.

Se puede abrir más ventanas del chat del cliente para conectar más usuarios, por medio de la IP y del puerto se pueden conectar usuarios de otros computadores siempre y cuando estén conectados a la misma red del servidor y los otros clientes.

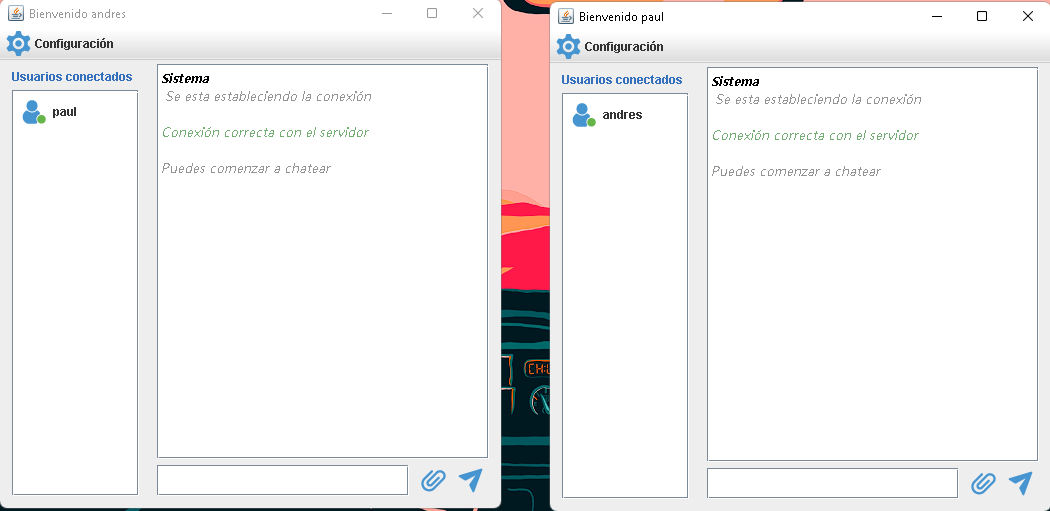


Figura 58. Ejecución de varios clientes.

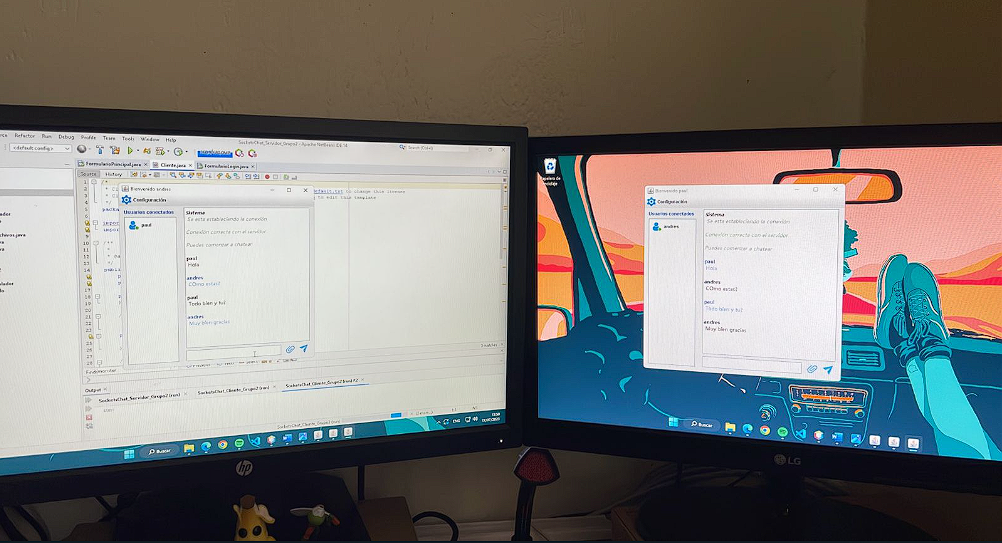


Figura 59. Ejecución del cliente en diferentes computadoras.

Para enviar un archivo se debe dar clic en el ícono similar a un clip a un lado del botón de enviar mensaje el cual desplegará la ventana del formulario de envió de archivos, dentro de este se selecciona el archivo y el destinatario.

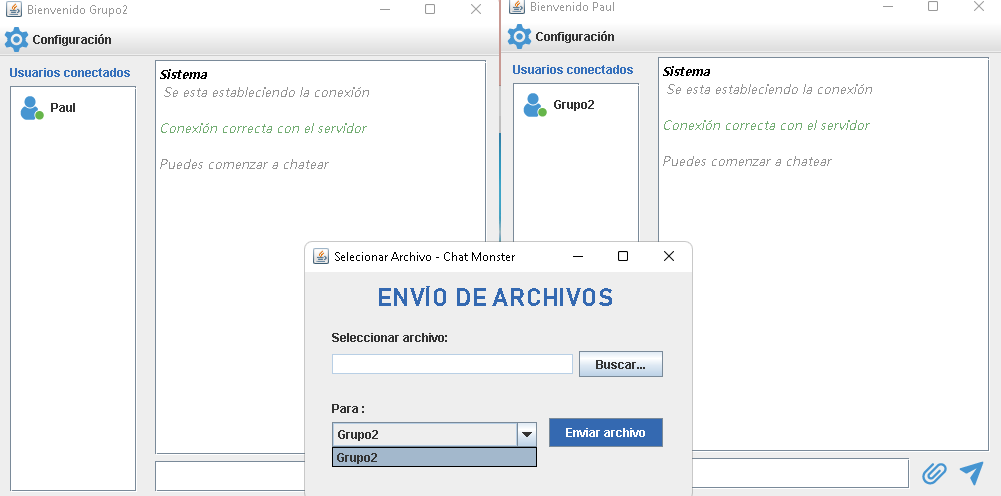


Figura 60. Envío de archivos.

# CONCLUSIONES

* Los Hilos en java nos permite generar programas capaces de realizar múltiples tareas al mismo tiempo que pueden aprovechar todas las capacidades de procesamiento de un computador.
* Java nos permite generar proyectos con la capacidad de utilizar hilos de manera fácil y rápida ya que incluye métodos y bibliotecas propias.
* Cuando se utilizan hilos las capacidades de nuestros proyectos aumentan ya que pueden hacer más tareas a la vez.
* El manejo de sockets permite comunicarnos a través de los puertos de la capa de transporte del modelo OSI (TCP/UDP) consiente transferir texto y archivos dentro de una misma red por lo que genera un alcance de aplicación demasiado extenso en el desarrollo de aplicaciones.
* El uso del modelo Cliente-Servidor facilita la conexión e interacción de los demás usuarios para una interacción múltiple dentro de la aplicación.

# RECOMENDACIONES

* Se recomienda renombrar el paquete principal del proyecto ya que esto nos ayuda a tener una mejor organización de nuestro código.
* Utilizar el modelo MVC ya que de esta manera podemos realizar modificaciones de nuestros programas sin la necesidad de cambiar todo el código.
* Se recomienda tener bases solidas del funcionamiento y manejo de hilos en Java.

# BIBLIOGRAFIA

|  |  |
| --- | --- |
| [1] | ORACLE CORPORATION, «JAVA INTRODUCCION,» ORACLE CORPORATION, 2017. [En línea]. Available: https://aws.amazon.com/es/what-is/java/. [Último acceso: 05 11 2022]. |
| [2] | Netbeans, «Welcome to Apache NetBeans,» Netbeans, [En línea]. Available: https://netbeans.apache.org/. [Último acceso: 05 11 2022]. |
| [3] | theastrology, «theastrology,» theastrology, 2018. [En línea]. Available: https://es.theastrologypage.com/concurrency. [Último acceso: 26 11 2022]. |
| [4] | EDTeam, «EDTeam,» EDTeam, 2018. [En línea]. Available: https://ed.team/blog/como-funcionan-los-hilos-en-programacion. [Último acceso: 26 11 2022]. |
| [5] | Universidad de Alicante, «Universidad de Alicante,» Universidad de Alicante, 2012. [En línea]. Available: http://www.jtech.ua.es/dadm/restringido/java/sesion05-apuntes.html#:~:text=En%20Java%20los%20hilos%20est%C3%A1n,definir%20el%20m%C3%A9todo%20run().. [Último acceso: 26 11 2022]. |
| [6] | Ilma, «NordVPN,» 13 11 2019. [En línea]. Available: https://nordvpn.com/es/blog/protocolo-tcp-udp/. [Último acceso: 1 12 2022]. |
| [7] | S. De Luz, «Redes Zone,» 16 10 2022. [En línea]. Available: https://www.redeszone.net/tutoriales/configuracion-puertos/que-es-socket-tcp-udp-diferencias-puertos/. [Último acceso: 1 12 2022]. |
| [8] | D. Meza, «Programar ya,» 2021. [En línea]. Available: https://www.programarya.com/Cursos-Avanzados/Java/Sockets. [Último acceso: 1 12 2022]. |