**Instituto Politécnico Nacional.**

**Escuela Superior De Cómputo.**





**Materia:**

**Aplicaciones Para Comunicaciones en Red.**

**Tema:**

**Reporte**

**(Práctica 3)**

**Profesor:**

**Axel Ernesto Moreno Cervantes.**

**Alumnos:**

**Luis Enrique Rojas Alvarado.**

**Mario Alberto Miranda Sandoval.**

**Grupo:**

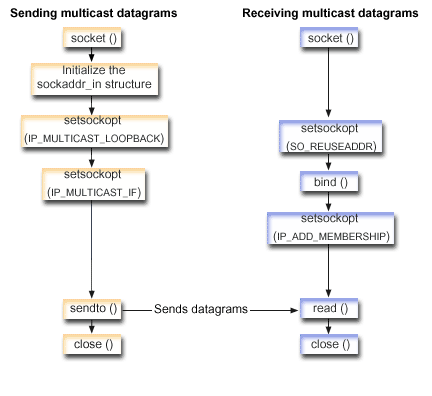
**3CM5.**

**Introducción.**

A veces nos interesa que un ordenador pueda enviar un mensaje por red y que este sea recibido por otros ordenadores simultáneamente. Para ello están las direcciones multicast.

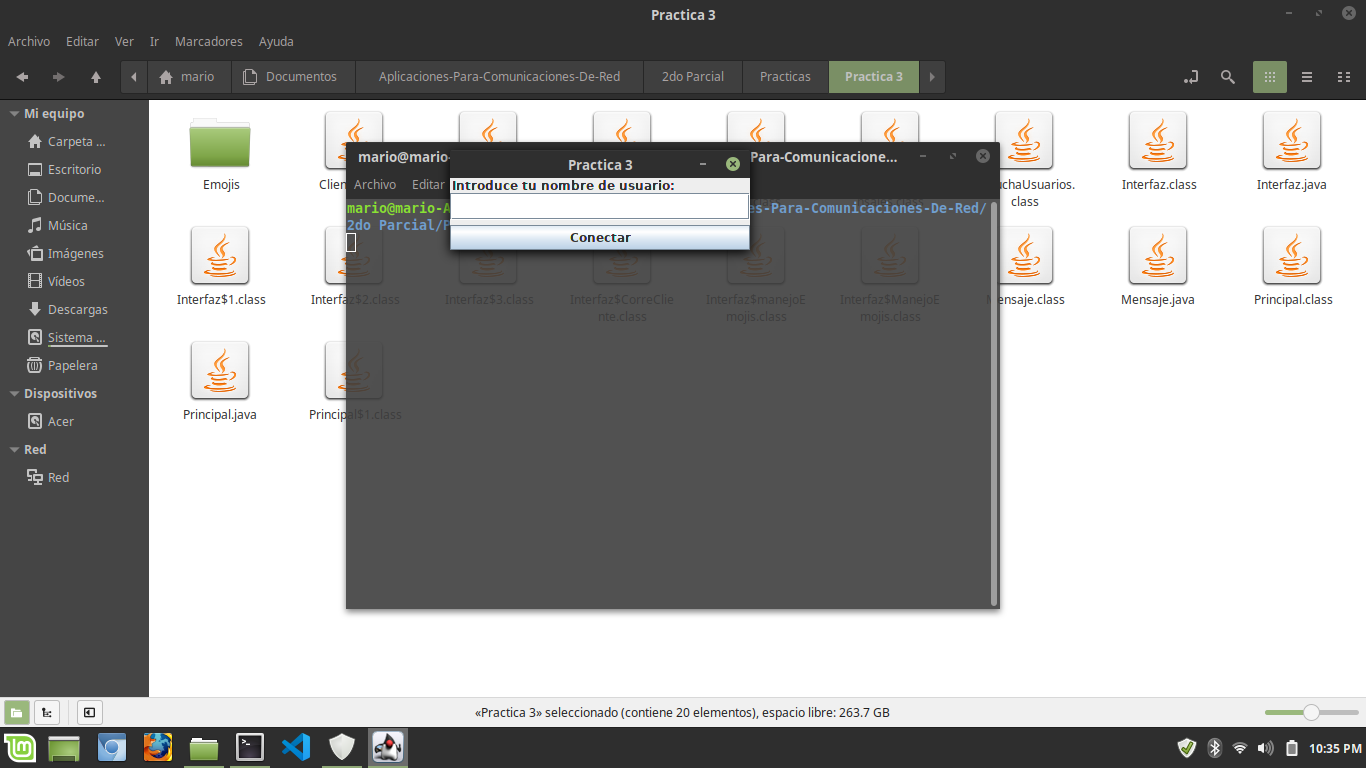
Son direcciones en el rango 224.0.0.0 a 239.255.255.255. La 224.0.0.0 está reservada y no puede usarse. Enviando mensajes por estas direcciones, cualquier otro ordenador en la red que las escuche podría leer dicho mensaje, independientemente de cuál sea la IP real de ese ordenador.

Es decir, si un ordenador quiere enviar un mensaje simultáneamente a varios, puede hacerlo enviando el mensaje a una de estas IPs, los demás ordenadores deben estar a la escucha de dichas IPs para recibir el mensaje.

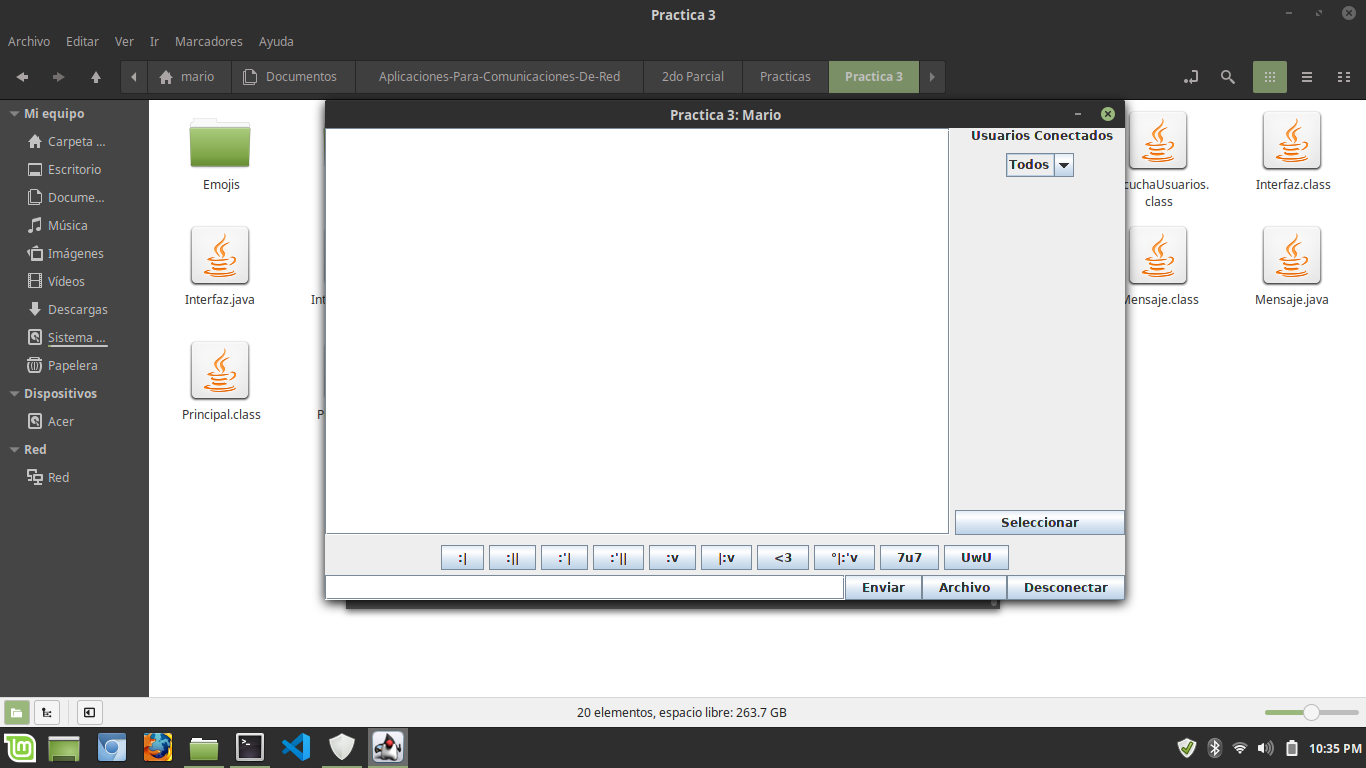


**Desarrollo.**

Para el desarrollo de esta práctica se genero los siguiente, primero una interfaz donde se coloca el nombre del usuario, mientras que la segunda es el chat principal.



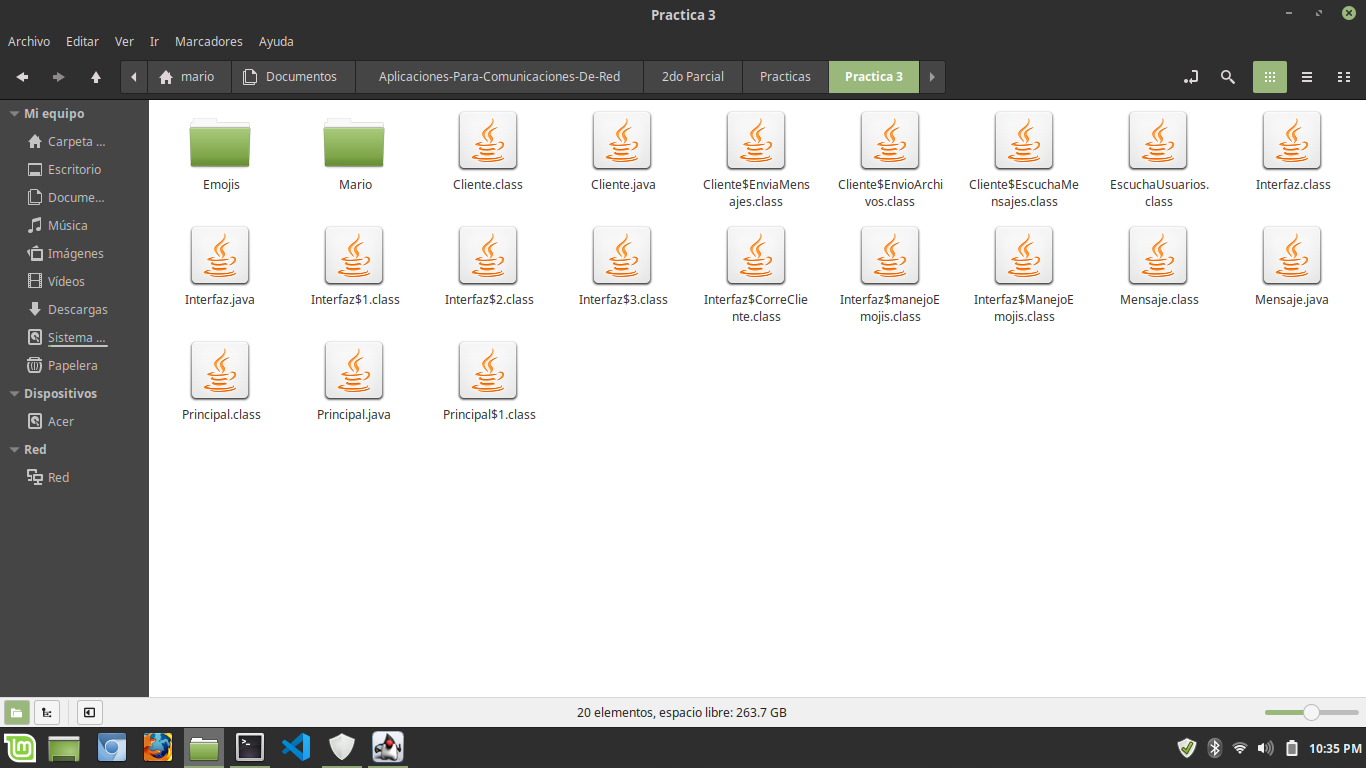
Interfaz de introducción de nombre.



Interfaz del chat.

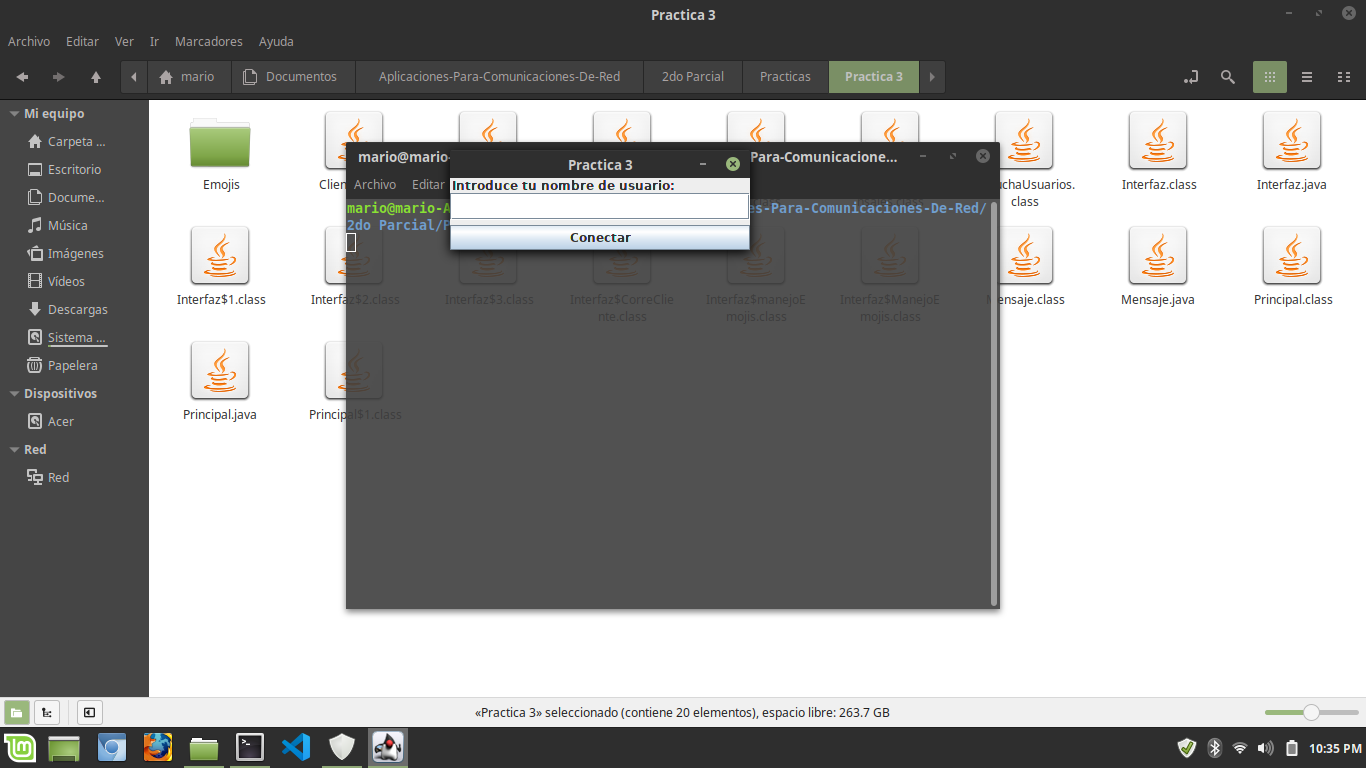
Una vez dado esto el usuario está en posición de entablar conversaciones en la sala, mandar y recibir archivos, y por último mandar mensajes privados.

Una consideración más es, debido al control de los usuarios el programa tiene la facultad de crear carpetas de usuario de acuerdo al nombre de este.

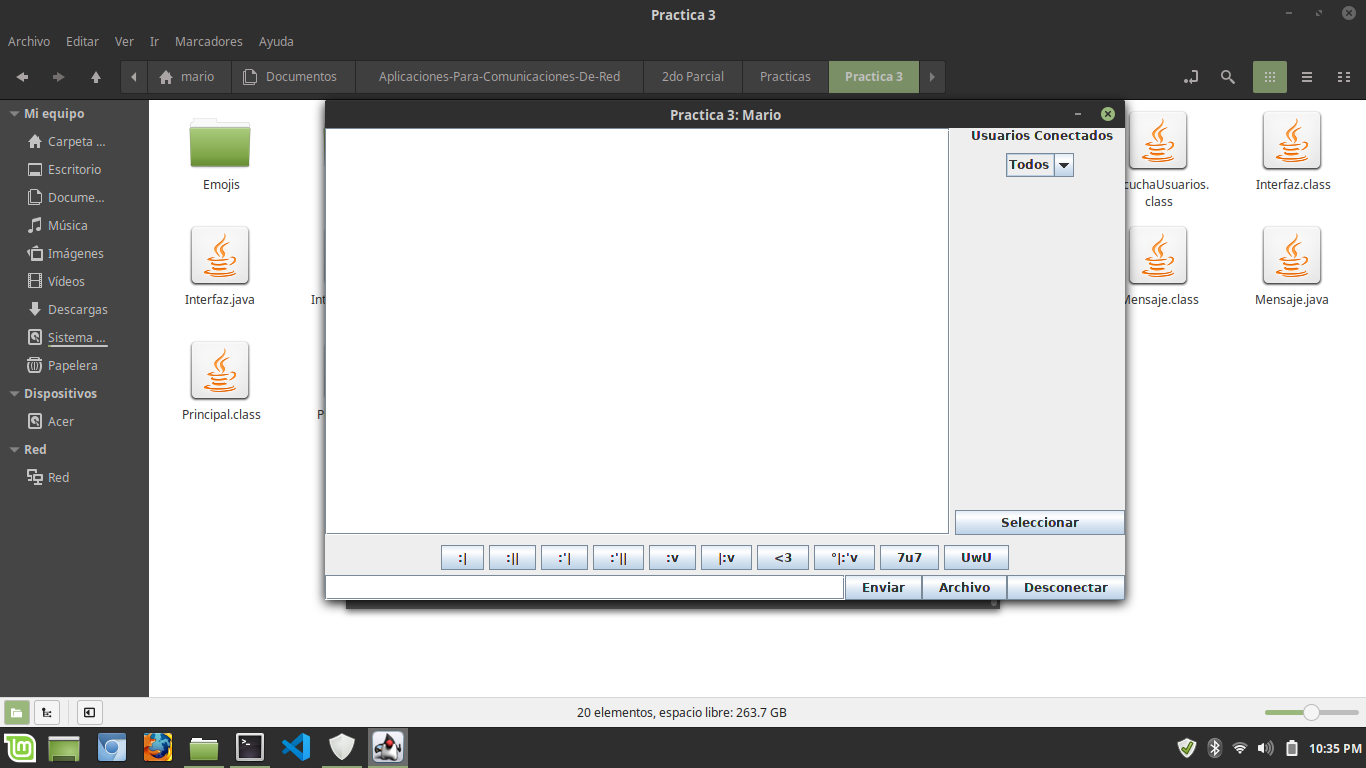


Carpeta Creada para el usuario **Mario**.

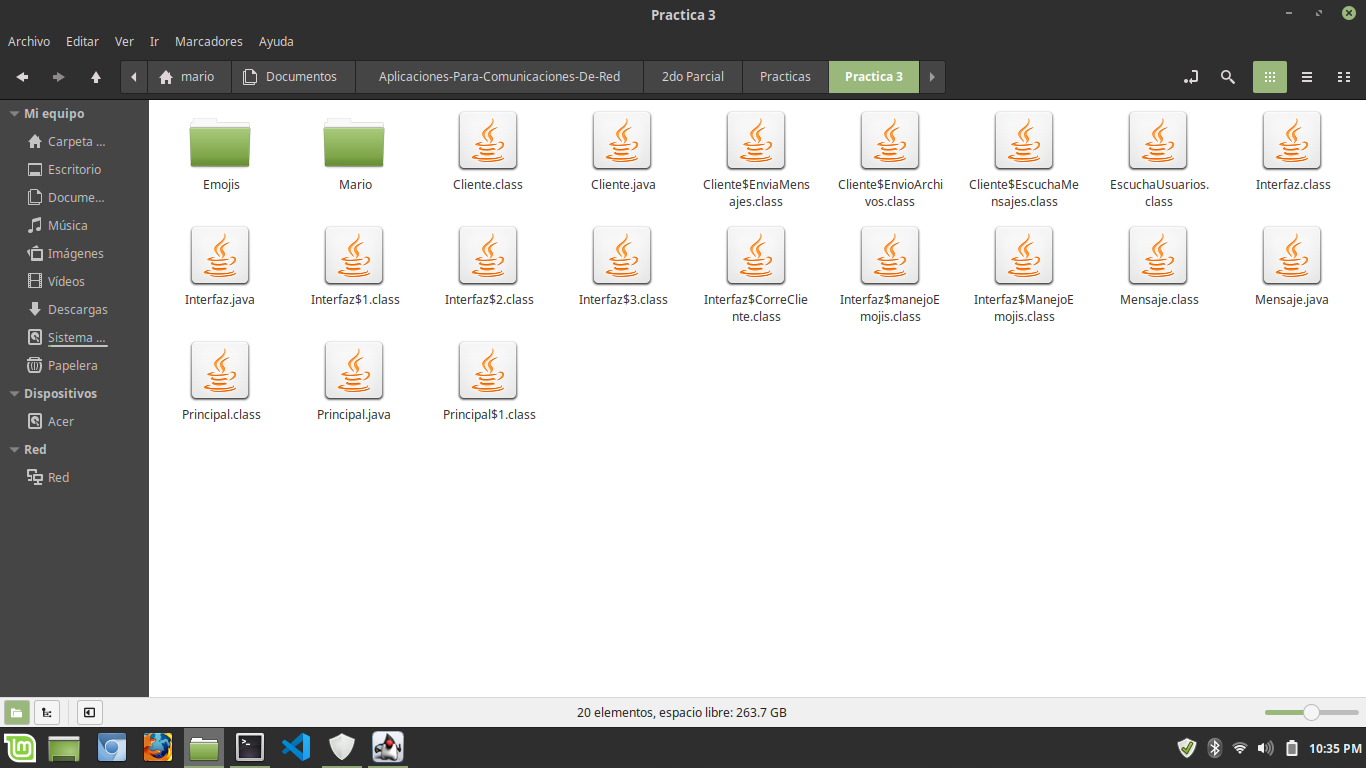
**Pruebas.**



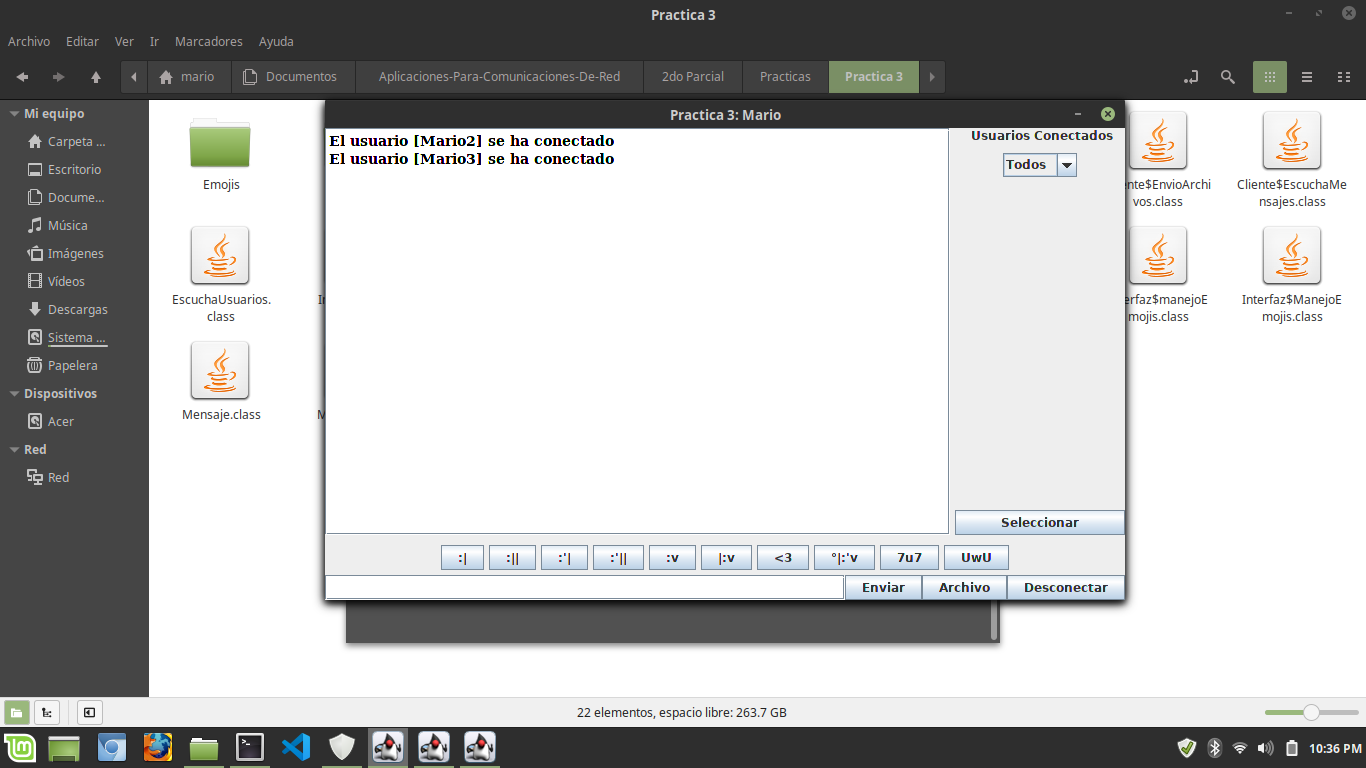
Usuario Registrado.



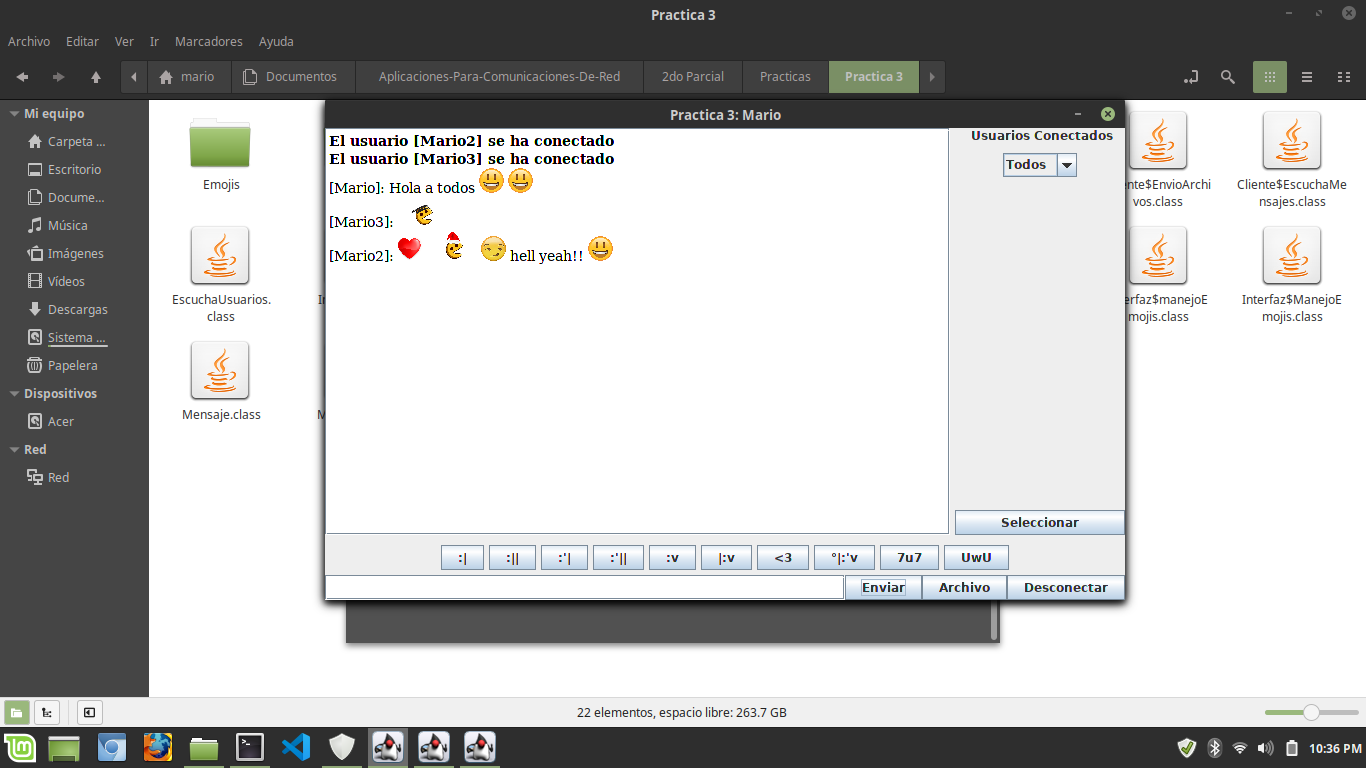
Interfaz de usuario principal.



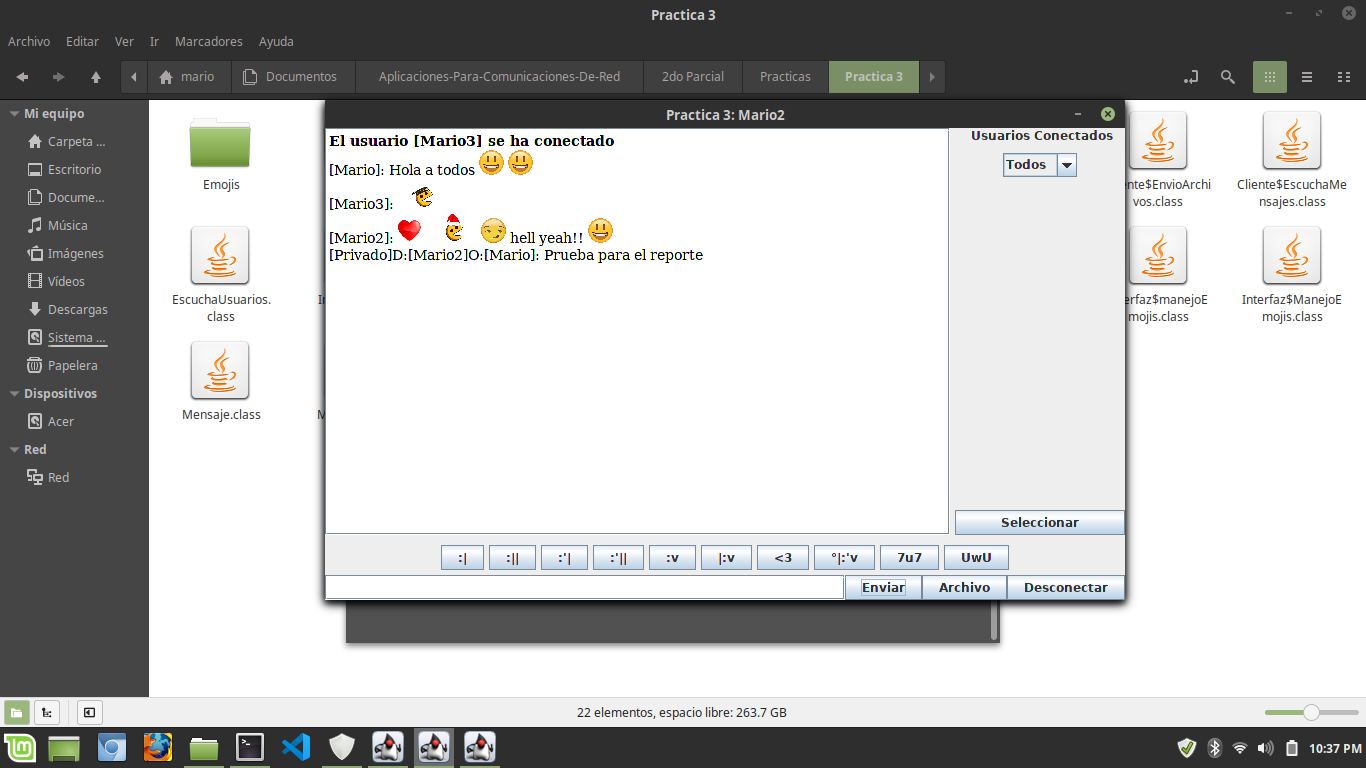
Carpeta de usuario creada.



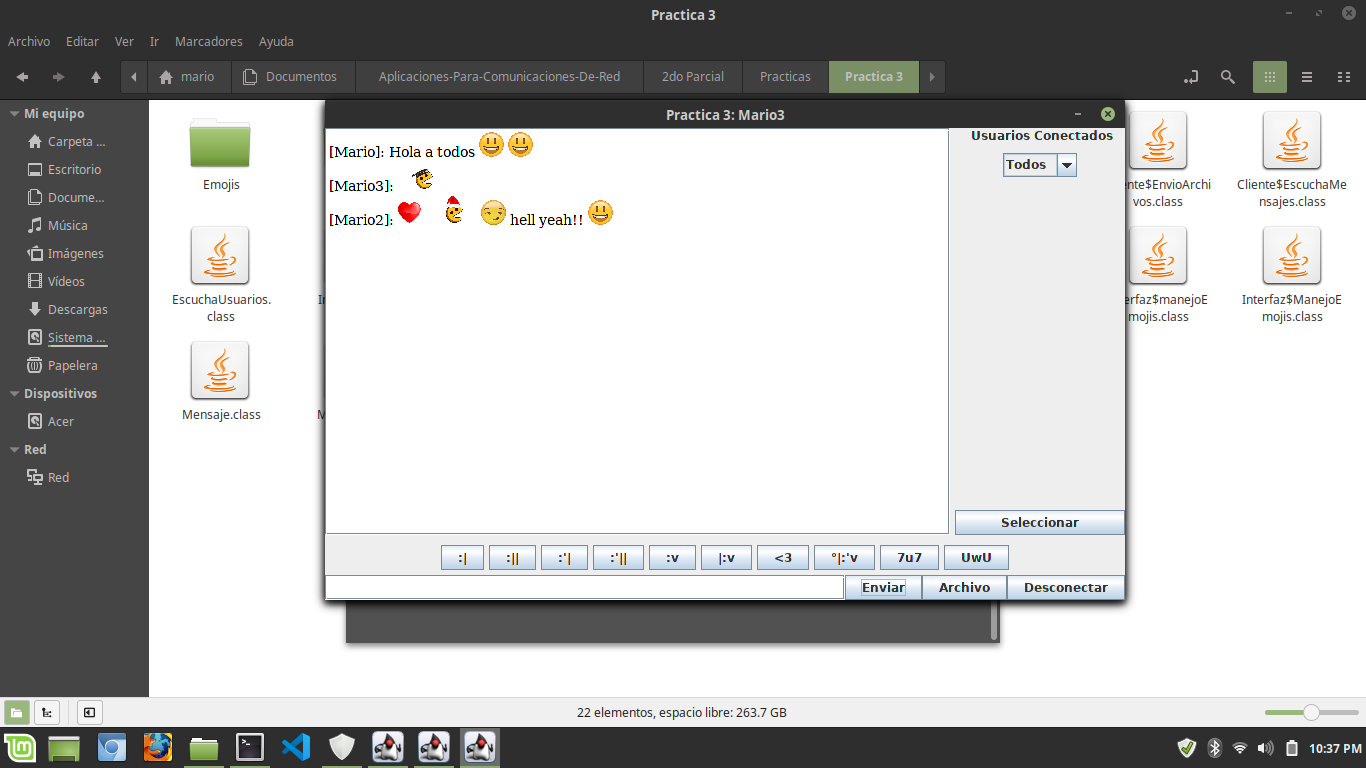
Anuncio de nuevos usuarios.



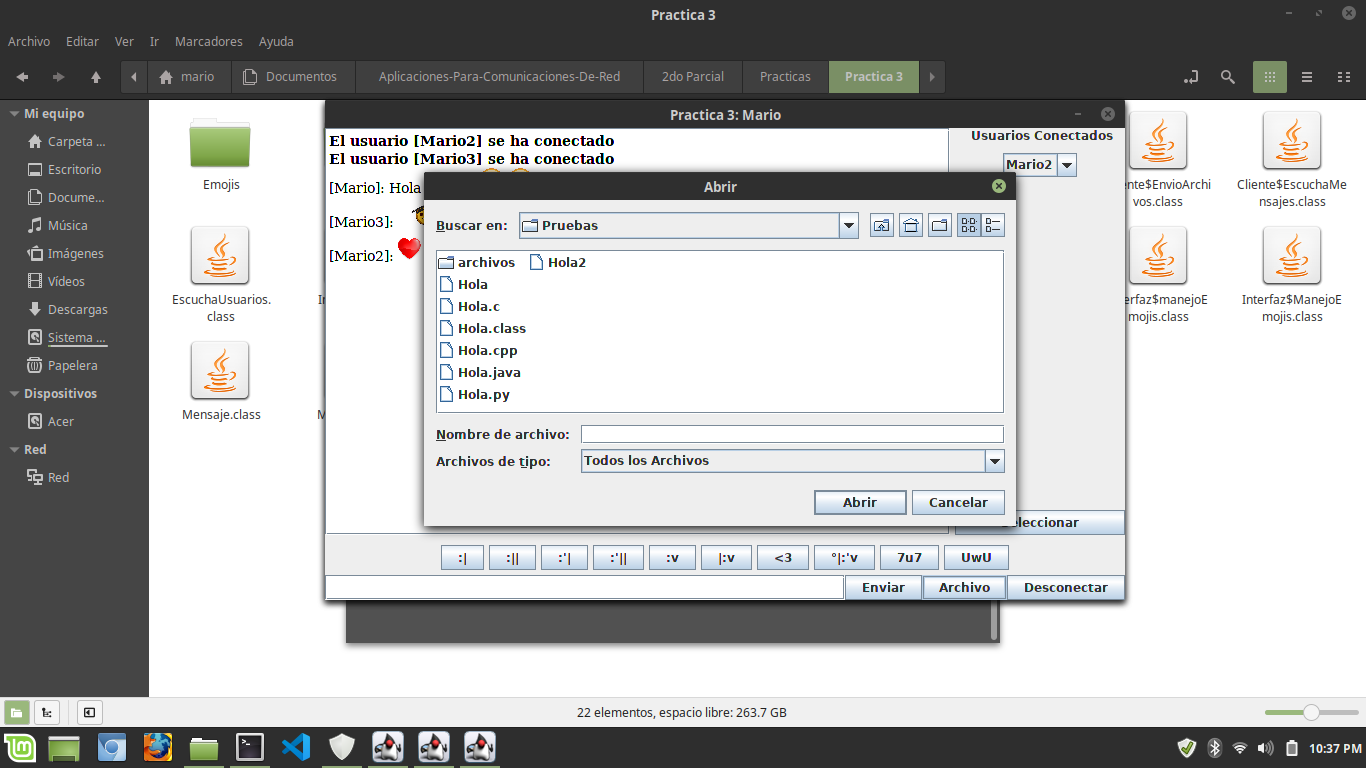
Envió de mensajes.



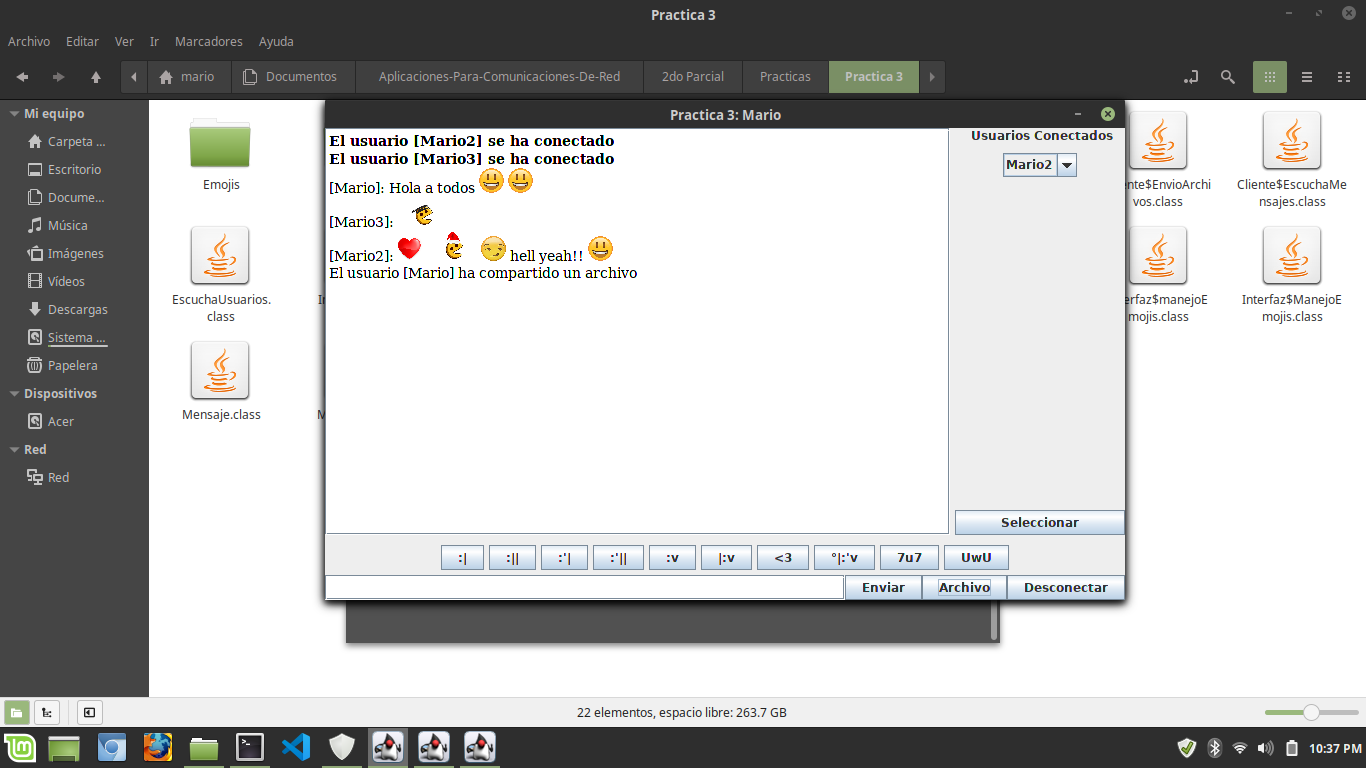
Envió de mensajes privados.



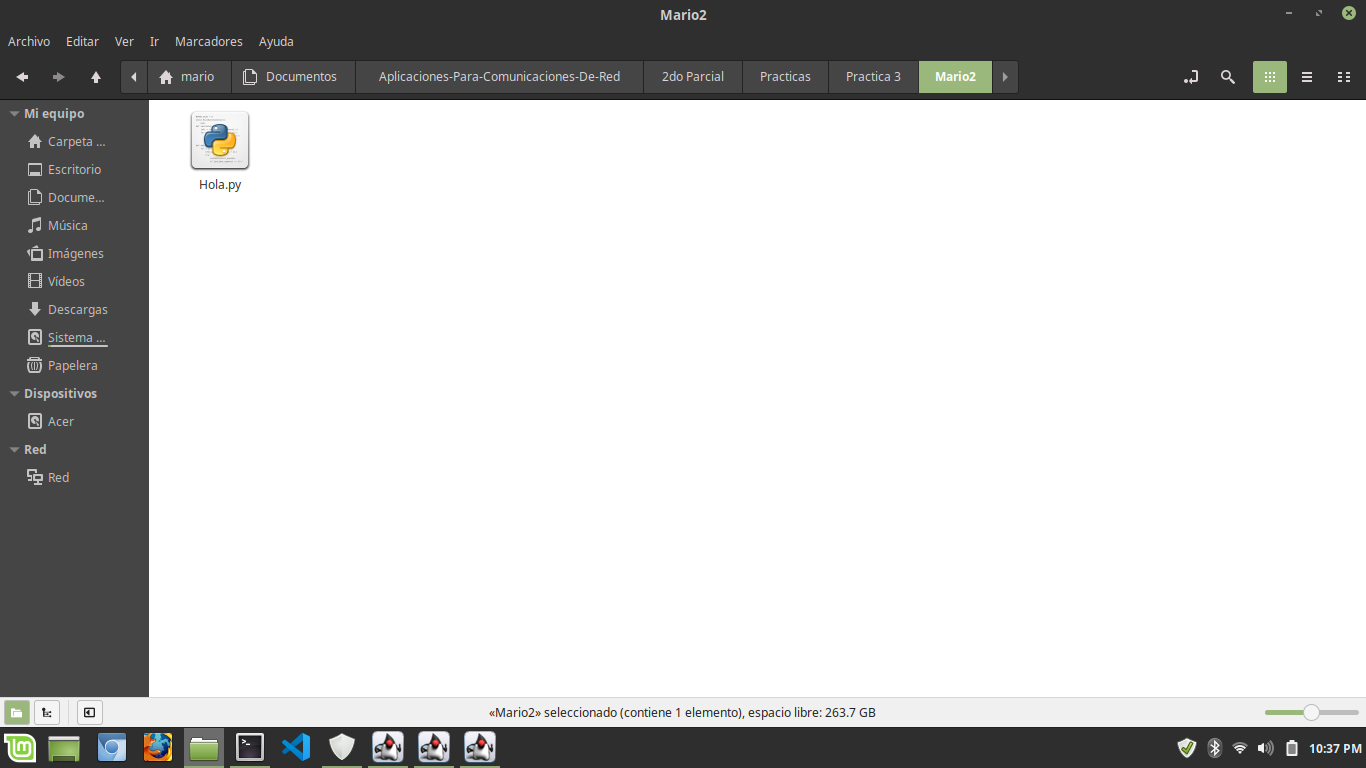
Comprobación de mensajes privados.

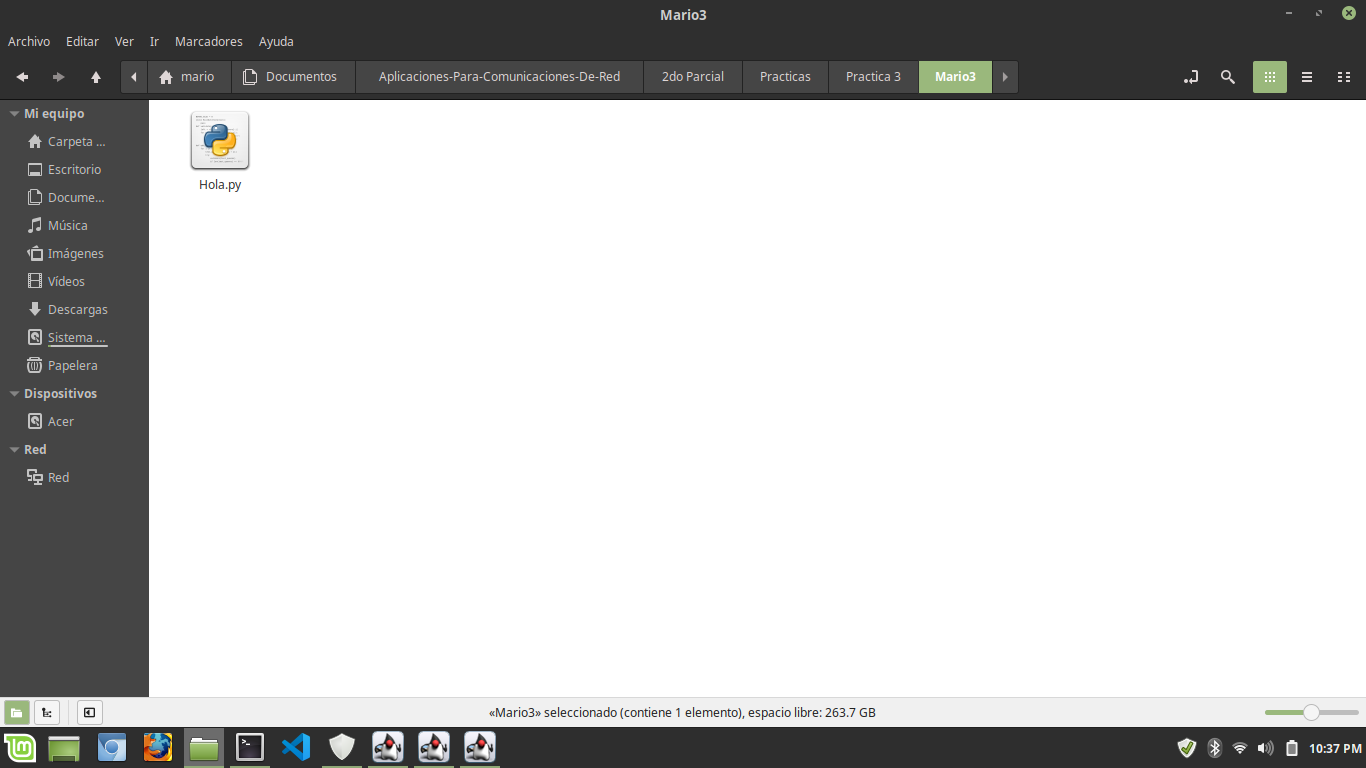


Enviando Archivos.



Anuncio de envió de archivos.





Comprobación del envío de archivos.

**Código.**

**Principal.java**

1. **import** javax.swing.\*;
2. **import** java.awt.BorderLayout;
4. **public** **class** Principal **extends** JFrame {
5. **private** **static** **final** **long** serialVersionUID = 1L;
7. **public** Principal() {
8. setBounds(450, 150, 300, 100);
9. setResizable(**false**);
10. setTitle("Practica 3");
12. panelPrincipal = **new** JPanel();
13. panelCentral = **new** JPanel();
14. panelPrincipal.setLayout(**new** BorderLayout(5, 5));
15. panelCentral.setLayout(**new** BoxLayout(**this**.panelCentral, BoxLayout.Y\_AXIS));
16. nombreUsuario = **new** JLabel("Introduce tu nombre de usuario:");
17. campoUsuario = **new** JTextField(30);
18. botonConectar = **new** JButton("Conectar");
20. botonConectar.addActionListener(**new** java.awt.event.ActionListener() {
21. **public** **void** actionPerformed(java.awt.event.ActionEvent e) {
22. **if** (!campoUsuario.getText().equals("")) {
23. setVisible(**false**);
24. **new** Interfaz(HOST, PUERTO, campoUsuario.getText().trim());
25. } **else**
26. JOptionPane.showMessageDialog(Principal.**this**, "Nombre de usuario Vacio", "Error", JOptionPane.ERROR\_MESSAGE);
27. }
28. });
30. panelCentral.add(nombreUsuario);
31. panelCentral.add(campoUsuario);
32. panelPrincipal.add(panelCentral, BorderLayout.CENTER);
33. panelPrincipal.add(botonConectar, BorderLayout.SOUTH);
34. add(panelPrincipal);
36. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
37. setVisible(**true**);
38. }
40. **public** **static** **void** main(String[] args) {
41. **new** Principal();
42. }
44. **private** **final** String HOST = "230.1.1.1";
45. **private** **final** **int** PUERTO = 9000;
46. **private** JPanel panelPrincipal;
47. **private** JPanel panelCentral;
48. **private** JLabel nombreUsuario;
49. **private** JTextField campoUsuario;
50. **private** JButton botonConectar;
51. }

**Interfaz.java**

1. **import** javax.swing.\*;
2. **import** java.awt.\*;
3. **import** java.awt.event.\*;
5. **public** **class** Interfaz **extends** JFrame {
6. **private** **static** **final** **long** serialVersionUID = 2L;
8. **public** Interfaz(String host, **int** puerto, String nombre) {
9. // -----------------Recibiendo Parametros------------------//
10. **this**.host = host;
11. **this**.puerto = puerto;
12. **this**.nombre = nombre;
13. // ----------------Creando Interfaz-------------------------//
14. setBounds(325, 100, 800, 500);
15. setTitle("Practica 3: " + nombre);
16. setResizable(**false**);
18. panelPrincipal = **new** JPanel();
19. panelCentral = **new** JPanel();
20. panelInferior = **new** JPanel();
21. panelEmojis = **new** JPanel();
22. panelFunciones = **new** JPanel();
23. panelUsuarios = **new** JPanel();
24. panelCombo = **new** JPanel();
25. editor = **new** JEditorPane("text/html", **null**);
26. editor.setEditable(**false**);
27. areaMensaje = **new** JTextArea();
28. areaMensaje.setLineWrap(**true**);
29. botonesEmojis = **new** JButton[textoBotonesEmojis.length];
30. enviar = **new** JButton("Enviar");
31. archivo = **new** JButton("Archivo");
32. desconectar = **new** JButton("Desconectar");
33. seleccion = **new** JButton("Seleccionar");
34. usuariosConectados = **new** JLabel("    Usuarios Conectados   ");
35. escuchaEmojis = **new** ManejoEmojis();
36. usuarioConectado = **new** JComboBox<>();
37. usuarioConectado.addItem("Todos");
39. enviar.addActionListener(**new** ActionListener() {
40. **public** **void** actionPerformed(ActionEvent e) {
41. miCliente.enviar(**new** Mensaje("[" + nombre + "]: " + areaMensaje.getText(), nombre, "", 1));
42. areaMensaje.setText("");
43. }
44. });
46. archivo.addActionListener(**new** ActionListener() {
47. **public** **void** actionPerformed(ActionEvent e) {
48. JFileChooser jf = **new** JFileChooser();
49. jf.requestFocus();
50. **int** r = jf.showOpenDialog(Interfaz.**this**);
51. **if** (r == JFileChooser.APPROVE\_OPTION) {
52. miCliente.enviarArchivo(jf.getSelectedFile());
53. String mensaje = "El usuario [" + nombre + "] ha compartido un archivo";
54. miCliente.enviar(**new** Mensaje(mensaje, nombre, "", 1));
55. }
56. }
57. });
59. seleccion.addActionListener(**new** ActionListener() {
60. **public** **void** actionPerformed(ActionEvent e) {
61. miCliente.enviar(**new** Mensaje("[" + nombre + "]: " + areaMensaje.getText(), nombre,
62. (String) usuarioConectado.getSelectedItem(), 4));
63. areaMensaje.setText("");
64. }
65. });
67. panelPrincipal.setLayout(**new** BorderLayout(5, 5));
68. panelCentral.setLayout(**new** BorderLayout(5, 5));
69. panelInferior.setLayout(**new** BoxLayout(**this**.panelInferior, BoxLayout.Y\_AXIS));
70. panelFunciones.setLayout(**new** BoxLayout(**this**.panelFunciones, BoxLayout.X\_AXIS));
71. panelUsuarios.setLayout(**new** BorderLayout(5, 5));
73. colocarBotones();
74. addWindowListener(**new** CorreCliente());
75. panelCombo.add(usuarioConectado);
76. panelUsuarios.add(usuariosConectados, BorderLayout.NORTH);
77. usuariosConectados.setAlignmentX(SwingConstants.CENTER);
78. panelUsuarios.add(panelCombo, BorderLayout.CENTER);
79. panelUsuarios.add(seleccion, BorderLayout.SOUTH);
80. panelCentral.add(**new** JScrollPane(editor), BorderLayout.CENTER);
81. panelCentral.add(panelUsuarios, BorderLayout.EAST);
82. panelFunciones.add(**new** JScrollPane(areaMensaje));
83. panelFunciones.add(enviar);
84. panelFunciones.add(archivo);
85. panelFunciones.add(desconectar);
86. panelInferior.add(panelEmojis);
87. panelInferior.add(panelFunciones);
88. panelPrincipal.add(panelCentral, BorderLayout.CENTER);
89. panelPrincipal.add(panelInferior, BorderLayout.SOUTH);
90. add(panelPrincipal);
92. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
93. setVisible(**true**);
94. }
96. **private** **void** colocarBotones() {
97. **for**(**int** i = 0; i < botonesEmojis.length; i += 1) {
98. botonesEmojis[i] = **new** JButton("" + textoBotonesEmojis[i]);
99. botonesEmojis[i].addActionListener(escuchaEmojis);
100. panelEmojis.add(botonesEmojis[i]);
101. }
102. }
104. **private** **class** ManejoEmojis **implements** ActionListener {
105. **public** **void** actionPerformed(ActionEvent e) {
106. areaMensaje.append(" " + e.getActionCommand() + " ");
107. }
108. }
110. **private** **class** CorreCliente **extends** WindowAdapter {
111. **public** **void** windowOpened(WindowEvent we) {
112. System.out.println("Ventana abierta");
113. miCliente = **new** Cliente(nombre, host, puerto, editor, usuarioConectado);
114. miCliente.saludo(nombre);
115. }
116. }
118. **private** String host;
119. **private** **int** puerto;
120. **private** String nombre;
121. **private** JPanel panelPrincipal;
122. **private** JPanel panelCentral;
123. **private** JPanel panelInferior;
124. **private** JPanel panelEmojis;
125. **private** JPanel panelFunciones;
126. **private** JPanel panelUsuarios;
127. **private** JEditorPane editor;
128. **private** JButton[] botonesEmojis;
129. **private** String[] textoBotonesEmojis = {
130. ":|",
131. ":||",
132. ":'|",
133. ":'||",
134. ":v",
135. "|:v",
136. "<3",
137. "°|:'v",
138. "7u7",
139. "UwU"
140. };
141. **private** JTextArea areaMensaje;
142. **private** JButton enviar;
143. **private** JButton archivo;
144. **private** JButton desconectar;
145. **public** **static** JComboBox<String> usuarioConectado;
146. **private** JButton seleccion;
147. **private** JLabel usuariosConectados;
148. **private** ActionListener escuchaEmojis;
149. **private** Cliente miCliente;
150. **private** JPanel panelCombo;
151. }

**Mensaje.java**

1. **import** java.io.\*;
3. **public** **class** Mensaje **implements** Serializable {
4. **private** **static** **final** **long** serialVersionUID = 3L;
5. **public** Mensaje(String mensaje, String usuarioOrigen, String usuarioDestino, **int** tipo) {
6. **this**.mensaje = mensaje;
7. **this**.usuarioOrigen = usuarioOrigen;
8. **this**.usuarioDestino = usuarioDestino;
9. **this**.tipo = tipo;
10. }
12. **public** Mensaje(String nombreArchivo, String usuarioOrigen, String usuarioDestino, **int** tipo, **long** tamanio, String ruta, **int** np) {
13. **this**.nombreArchivo = nombreArchivo;
14. **this**.usuarioOrigen = usuarioOrigen;
15. **this**.usuarioDestino = usuarioDestino;
16. **this**.tamanio = tamanio;
17. **this**.ruta = ruta;
18. **this**.np = np;
19. **this**.tipo = tipo;
20. }
22. **public** String getMensaje() { **return** mensaje; }
23. **public** String getUsuarioOrigen() { **return** usuarioOrigen; }
24. **public** String getUsuarioDestino() { **return** usuarioDestino; }
25. **public** **int** getTipo() { **return** tipo; }
27. **public** **void** setMensaje(String mensaje) { **this**.mensaje = mensaje; }
28. **public** **void** setUsuarioOrigen(String usuarioOrigen) { **this**.usuarioOrigen = usuarioOrigen; }
29. **public** **void** setUsuarioDestino(String usuarioDestino) { **this**.usuarioDestino = usuarioDestino; }
30. **public** **void** setTipo(**int** tipo) { **this**.tipo = tipo; }
32. **public** String getNombre() { **return** nombreArchivo; }
33. **public** **long** getTamanio() { **return** tamanio; }
34. **public** String ruta() { **return** ruta; }
35. **public** **int** getNp() { **return** np; }
36. **public** **byte**[] getDatos() { **return** datos; }
37. **public** **int** getBytesEnviados() { **return** bytesEnviados; }
39. **public** **void** setDatos(**byte**[] datos) { **this**.datos = datos; }
40. **public** **void** setBytesEnviados(**int** bytesEnviados) { **this**.bytesEnviados = bytesEnviados; }
42. **private** String mensaje;
43. **private** String usuarioOrigen;
44. **private** String usuarioDestino;
45. **private** **int** tipo;
46. **private** String nombreArchivo;
47. **private** **long** tamanio;
48. **private** String ruta;
49. **private** **int** np;
50. **private** **byte**[] datos;
51. **private** **int** bytesEnviados;
52. }

**Cliente.java**

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **import** javax.swing.\*;
4. **import** javax.swing.text.html.HTMLEditorKit;
5. **import** java.util.\*;
6. **import** java.util.regex.\*;
8. **public** **class** Cliente {
9. **public** Cliente(String nombre, String host, **int** puerto, JEditorPane editor, JComboBox<String> usuarioConectado) {
10. **this**.nombre = nombre;
11. **this**.host = host;
12. **this**.puerto = puerto;
13. **this**.editor = editor;
14. **this**.usuarioConectado = usuarioConectado;
16. **try** {
17. cliente = **new** MulticastSocket(puerto);
18. grupo = InetAddress.getByName(host);
19. cliente.joinGroup(grupo);
20. } **catch** (Exception e) {
21. e.printStackTrace();
22. }
24. hiloEscucha = **new** EscuchaMensajes();
25. escucha = **new** Thread(hiloEscucha);
26. escucha.start();
27. carpeta();
28. }
30. **private** **class** EscuchaMensajes **implements** Runnable {
31. **public** **void** run() {
32. System.out.println("Escuchando Mensajes");
33. **try** {
34. DatagramPacket recibido = **new** DatagramPacket(**new** **byte**[6500], 6500);
35. **while** (**true**) {
36. cliente.receive(recibido);
37. ObjectInputStream ois = **new** ObjectInputStream(**new** ByteArrayInputStream(recibido.getData()));
38. Mensaje msj = (Mensaje) ois.readObject();
39. **if**(msj.getTipo() == 0 && !msj.getUsuarioOrigen().equals(nombre)) {
40. **byte**[] bmsj = msj.getMensaje().getBytes();
41. **byte**[] busuario = msj.getUsuarioOrigen().getBytes();
42. String mensaje = **new** String(bmsj, 0, msj.getMensaje().length());
43. String usuario = **new** String(busuario, 0, msj.getUsuarioOrigen().length());
44. HTMLEditorKit kit = (HTMLEditorKit) editor.getEditorKit();
45. StringReader reader = **new** StringReader(mensaje);
46. kit.read(reader, editor.getDocument(), editor.getDocument().getLength());
47. usuarioConectado.addItem(usuario);
48. ByteArrayOutputStream baos = **new** ByteArrayOutputStream();
49. ObjectOutputStream oos = **new** ObjectOutputStream(baos);
50. msj.setTipo(3);
51. msj.setUsuarioDestino(msj.getUsuarioOrigen());
52. msj.setUsuarioOrigen(nombre);
53. oos.writeObject(msj);
54. oos.flush();
55. **byte**[] b = baos.toByteArray();
56. DatagramPacket re = **new** DatagramPacket(b, b.length, grupo, puerto);
57. cliente.send(re);
58. } **else** **if**(msj.getTipo() == 1) {
59. **byte**[] bmsj = msj.getMensaje().getBytes();
60. String mensaje = **new** String(bmsj, 0, msj.getMensaje().length());
61. HTMLEditorKit kit = (HTMLEditorKit) editor.getEditorKit();
62. StringReader reader = **new** StringReader(mensaje);
63. kit.read(reader, editor.getDocument(), editor.getDocument().getLength());
64. } **else** **if**(msj.getTipo() == 2) {
65. **if**(!msj.getUsuarioOrigen().equals(nombre))
66. recibirArchivo(msj);
67. }**else** **if**(msj.getTipo() == 3 && !msj.getUsuarioOrigen().equals(nombre) && msj.getUsuarioDestino().equals(nombre)) {
68. **byte**[] busuario = msj.getUsuarioOrigen().getBytes();
69. String usuario = **new** String(busuario, 0, msj.getUsuarioOrigen().length());
70. usuarioConectado.addItem(usuario);
71. } **else** **if**(msj.getTipo() == 4 && msj.getUsuarioDestino().equals(nombre) && !msj.getUsuarioOrigen().equals(nombre)) {
72. **byte**[] bmsj = msj.getMensaje().getBytes();
73. String mensaje = **new** String(bmsj, 0, msj.getMensaje().length());
74. mensaje = "[Privado]D:[" + nombre + "]O:" + mensaje;
75. HTMLEditorKit kit = (HTMLEditorKit) editor.getEditorKit();
76. StringReader reader = **new** StringReader(mensaje);
77. kit.read(reader, editor.getDocument(), editor.getDocument().getLength());
78. }
79. ois.close();
80. }
81. } **catch** (Exception e) {
82. e.printStackTrace();
83. }
84. }
85. }
87. **private** **class** EnviaMensajes **implements** Runnable {
88. **private** EnviaMensajes(Mensaje msj) {
89. **this**.msj = msj;
90. }
92. **public** **void** run() {
93. // editor.setText("<b>Enviando Mensajes</b>" + (++i));
94. **try** {
95. ByteArrayOutputStream baos = **new** ByteArrayOutputStream();
96. ObjectOutputStream oos = **new** ObjectOutputStream(baos);
97. oos.writeObject(msj);
98. oos.flush();
99. **byte**[] msj = baos.toByteArray();
100. DatagramPacket p = **new** DatagramPacket(msj, msj.length, grupo, puerto);
101. cliente.send(p);
102. oos.close();
103. baos.close();
104. } **catch** (IOException e) {
105. e.printStackTrace();
106. }
107. }
109. **private** Mensaje msj;
110. }
112. **public** **void** enviar(Mensaje msj) {
113. p = Pattern.compile("UwU");
114. m = p.matcher(msj.getMensaje());
116. **if**(m.find()) {
117. System.out.println("Se encontro UwU");
118. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/uwu.png").toString();
119. String img = "<img src = '" + imgsrc + "' width = 50 height = 50>";
120. String cad = msj.getMensaje().replace("UwU", img);
121. msj.setMensaje(cad);
122. }
124. p = Pattern.compile("7u7");
125. m = p.matcher(msj.getMensaje());
127. **if**(m.find()) {
128. System.out.println("Se encontro 7u7");
129. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/7u7.png").toString();
130. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
131. String cad = msj.getMensaje().replace("7u7", img);
132. msj.setMensaje(cad);
133. }
135. p = Pattern.compile("°|:'v");
136. m = p.matcher(msj.getMensaje());
138. **if**(m.find()) {
139. System.out.println("Se encontro °<:{v");
140. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/navidad.png").toString();
141. String img = "<img src = '" + imgsrc + "' width = 50 height = 30>";
142. String cad = msj.getMensaje().replace("°|:'v", img);
143. msj.setMensaje(cad);
144. }
146. p = Pattern.compile("<3");
147. m = p.matcher(msj.getMensaje());
149. **if**(m.find()) {
150. System.out.println("Se encontro <3");
151. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/cora.png").toString();
152. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
153. String cad = msj.getMensaje().replace("<3", img);
154. msj.setMensaje(cad);
155. }
157. p = Pattern.compile("|:v");
158. m = p.matcher(msj.getMensaje());
160. **if**(m.find()) {
161. System.out.println("Se encontro {:v");
162. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/jackie.png").toString();
163. String img = "<img src = '" + imgsrc + "' width = 50 height = 30>";
164. String cad = msj.getMensaje().replace("|:v", img);
165. msj.setMensaje(cad);
166. }
168. p = Pattern.compile(":v");
169. m = p.matcher(msj.getMensaje());
171. **if**(m.find()) {
172. System.out.println("Se encontro :v");
173. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/pacman.png").toString();
174. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
175. String cad = msj.getMensaje().replace(":v", img);
176. msj.setMensaje(cad);
177. }
179. p = Pattern.compile(":'||");
180. m = p.matcher(msj.getMensaje());
182. **if**(m.find()) {
183. System.out.println("Se encontro :'(");
184. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/lagrima.png").toString();
185. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
186. String cad = msj.getMensaje().replace(":'||", img);
187. msj.setMensaje(cad);
188. }
190. p = Pattern.compile(":'|");
191. m = p.matcher(msj.getMensaje());
193. **if**(m.find()) {
194. System.out.println("Se encontro :')");
195. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/risas.png").toString();
196. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
197. String cad = msj.getMensaje().replace(":'|", img);
198. msj.setMensaje(cad);
199. }
201. p = Pattern.compile(":||");
202. m = p.matcher(msj.getMensaje());
204. **if**(m.find()) {
205. System.out.println("Se encontro :(");
206. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/triste.png").toString();
207. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
208. String cad = msj.getMensaje().replace(":||", img);
209. msj.setMensaje(cad);
210. }
212. p = Pattern.compile(":|");
213. m = p.matcher(msj.getMensaje());
215. **if**(m.find()) {
216. System.out.println("Se encontro :)");
217. String imgsrc = Cliente.**class**.getClassLoader().getSystemResource("./Emojis/feliz.png").toString();
218. String img = "<img src = '" + imgsrc + "' width = 25 height = 25>";
219. String cad = msj.getMensaje().replace(":|", img);
220. msj.setMensaje(cad);
221. }
223. **new** Thread(**new** EnviaMensajes(msj)).start();
224. }
226. **public** **void** saludo(String nombre) {
227. String a = " <b>El usuario [" + nombre + "] se ha conectado</b>";
228. Mensaje m = **new** Mensaje(a, nombre, "", 0);
229. **new** Thread(**new** EnviaMensajes(m)).start();
230. }
232. **public** **void** carpeta() {
233. File carpeta = **new** File("./" + nombre);
234. **if**(!carpeta.exists()) {
235. **try** {
236. **if**(carpeta.mkdir())
237. System.out.println("Se creo la carpeta");
238. **else**
239. System.out.println("No se creo la carpeta");
240. }**catch**(SecurityException se) {
241. se.printStackTrace();
242. }
243. }
244. }
246. **private** **class** EnvioArchivos **implements** Runnable {
247. **private** EnvioArchivos(File file) {
248. **this**.file = file;
249. }
251. **public** **void** run() {
252. **try** {
253. DataInputStream dis = **new** DataInputStream(**new** FileInputStream(file));
254. **long** tamanio = dis.available();
255. **long** enviado = 0;
256. **int** n = 0;
257. **int** i = 0;
258. **while** (enviado < tamanio) {
259. Mensaje datos = **new** Mensaje(file.getName(), nombre, "", 2, file.length(), "", ++i);
260. ByteArrayOutputStream baos = **new** ByteArrayOutputStream(6400);
261. ObjectOutputStream oos = **new** ObjectOutputStream(**new** BufferedOutputStream(baos));
262. oos.flush();
263. **byte**[] b = **new** **byte**[4000];
264. n = dis.read(b);
265. **byte**[] b2 = **new** **byte**[n];
266. System.arraycopy(b, 0, b2, 0, n);
267. datos.setDatos(b2);
268. datos.setBytesEnviados(n);
269. oos.writeObject(datos);
270. oos.flush();
271. **byte**[] d = baos.toByteArray();
272. DatagramPacket paqueteEnvio = **new** DatagramPacket(d, d.length, grupo, puerto);
273. cliente.send(paqueteEnvio);
274. **try** {
275. Thread.sleep(500);
276. }**catch** (Exception e) { e.printStackTrace(); }
277. System.out.println("Numero paquete:" + i);
278. enviado += n;
279. oos.close();
280. baos.close();
281. }
282. **byte**[] bFinal = {0x02};
283. Mensaje paqueteFinal = **new** Mensaje(file.getName(), nombre, "", 2, file.length(), "", 0);
285. paqueteFinal.setDatos(bFinal);
286. ByteArrayOutputStream baos = **new** ByteArrayOutputStream();
287. ObjectOutputStream oos = **new** ObjectOutputStream(baos);
289. oos.writeObject(paqueteFinal);
290. oos.flush();
292. **byte**[] mnsj = baos.toByteArray();
294. DatagramPacket dp = **new** DatagramPacket(mnsj,mnsj.length, grupo, puerto);
295. cliente.send(dp);
297. System.out.println("Archivo Enviado");
299. oos.close();
300. baos.close();
301. //cliente.close();
302. dis.close();
303. } **catch**(Exception e) {
304. e.printStackTrace();
305. }//try/catch
306. }
308. **private** File file;
309. }
311. **public** **void** enviarArchivo(File file) {
312. **new** Thread(**new** EnvioArchivos(file)).start();
313. }
315. **private** **void** recibirArchivo(Mensaje datos) {
316. **try**{
317. System.out.println("Numero de paquete: " + datos.getNp());
318. **if**(datos.getNp() == 0) {
319. dos = **new** DataOutputStream(**new** FileOutputStream("./" + nombre + "/" + nombreArchivo));
320. **for**(**int** i = 0; i < lista.size(); i++) {
321. dos.write(lista.get(i));
322. }
323. dos.close();
324. lista.clear();
325. } **else** **if**(datos.getNp() == 1) {
326. lista = **new** ArrayList<>();
327. nombreArchivo = datos.getNombre();
328. lista.add(datos.getDatos());
329. } **else** {
330. **if**(nombreArchivo.equals(datos.getNombre()))
331. lista.add(datos.getDatos());
332. }
333. Thread.sleep(500);
334. }**catch**(Exception e) {
335. e.printStackTrace();
336. }
338. }
340. **private** String nombre;
341. **private** String nombreArchivo;
342. **private** String host;
343. **private** **int** puerto;
344. **private** JEditorPane editor;
345. **private** Thread escucha;
346. **private** Runnable hiloEscucha;
347. **private** MulticastSocket cliente;
348. **private** InetAddress grupo;
349. **private** JComboBox<String> usuarioConectado;
350. **private** DataOutputStream dos;
351. **private** ArrayList <**byte**[]> lista = **null**;
352. **private** Pattern p;
353. **private** Matcher m;
354. }

**Dificultades Encontradas.**

El primer punto a superar fue el comprender de buena manera como funcionaban los sockets multicast, ya que en un principio se tenia la vaga idea de usar un cliente y servidor, pero eso no era fácil de usar.

Una vez pasado este punto el siguiente, fue como actualizar la lista de contactos ya que en un punto pasaba de que se agregaban a modo que llegaban, es decir el primer usuario registrado es el que tenia la lista completa de todos los que se conectaban, el truco estuvo en poner un segundo anuncio que mandaba el nombre de usuario ya existente al recibir uno nuevo.

Otro punto a resaltar, fue el hecho de sobrecargar el constructor del objeto Mensaje, con el propósito de poder mandar mensajes y archivos sin problemas entre el envío de mensajes y archivos.

Por último, el uso de hilos fue imprescindible para evitar que la interfaz se quedara a la espera del final del proceso para poder seguir funcionando, el uso de clases internas fue esencial para poder usar diversos hilos con diferentes usos.

Como posibles mejoras existen demasiadas, como el hecho de hacer anuncios de los usuarios ya conectados aparte de añadirlos a la lista de contactos, validaciones de cuando el usuario ya esta conectado no se pueda registrar otro con el mismo nombre y quitar los usuarios de la lista de contactos cuando se desconecta.

**Conclusiones.**

**Luis Enrique Rojas Alvarado.**

En esta práctica pudimos observar las diferencias entre los sockets multicas y los de flujo. Pero la versatilidad que nos dan es amplia para hacer aplicaciones que deben tratar varios elementos interactuando entre sí, pero cuidando la integridad de los datos que se van a mandar, puesto que con UDP hay que buscar la forma de garantizar el envío de paquetes y el uso de hilos para soportar la concurrencia de múltiples usuarios al mismo tiempo.

**Miranda Sandoval Mario Alberto.**

Esta práctica fue interesante por el hecho de tener que usar los hilos y usar sockets multicast, el uso de clases privadas que implementaban la interfaz Runnable permitieron hacer que cada hilo actuara de manera diferente, mientras que usar distintos métodos para las llamadas dieron el éxito.

Para los emojis hubo que cambiar algunos caracteres del texto del botón, ya que el analizador de expresiones regulares de Java devolvía errores al usar paréntesis o llaves.