```
 1: import java.applet.*;
 2: import java.awt.*;
 3: import java.awt.event.*;
 4: import java.io.*;
 5: import java.net.*;
 6:
 7: /**
 8:  * <p>
 9:  * Simple Chat Room Applet.
10:  * Writing a Chat Room seems to be one of many obligatory rites (or wrongs)
11:  * of passage for Java experts these days.</p>
12:  * <p>
13:  * This one is a toy because it doesn't have much of a protocol, which
14:  * means we can't query the server as to * who's logged in,
15:  * or anything fancy like that. However, it works OK for small groups.</p>
16:  * <p>
17:  * Uses client socket w/ two Threads (main and one constructed),
18:  * one for reading and one for writing.</p>
19:  * <p>
20:  * Server multiplexes messages back to all clients.</p>
21:  * @author Ian Darwin
22:  * @version $Id: ChatRoom.java,v 1.4 2006/05/26 18:11:02 ian Exp $
23:  */
24: public class ChatRoom extends Applet {
25:
26:         private static final long serialVersionUID = -3686334002367908392L;
27:         /** Whether we are being run as an Applet or an Application */
28:         protected boolean inAnApplet = true;
29:         /** The state of logged-in-ness */
30:         protected boolean loggedIn;
31:         /* The Frame, for a pop-up, durable Chat Room. */
32:         protected Frame cp;
33:         /** The default port number */
34:         protected static final int PORTNUM = Chat.PORTNUM;
35:         /** The actual port number */
36:         protected int port;
37:         /** The network socket */
38:         protected Socket sock;
39:         /** BufferedReader for reading from socket */
40:         protected BufferedReader is;
41:         /** PrintWriter for sending lines on socket */
42:         protected PrintWriter pw;
43:         /** TextField for input */
44:         protected TextField tf;
```

```
45:         /** TextArea to display conversations */
46:         protected TextArea ta;
47:         /** The Login button */
48:         protected Button lib;
49:         /** The LogOUT button */
50:         protected Button lob;
51:         /** The TitleBar title */
52:         final static String TITLE = "Chat: Ian Darwin's Toy Chat Room Client";
53:         /** The message that we paint */
54:         protected String paintMessage;
55:
56:         /** init, overriding the version inherited from Applet */
57:         public void init() {
58:                 paintMessage = "Creating Window for Chat";
59:                 repaint();
60:                 cp = new Frame(TITLE);
61:                 cp.setLayout(new BorderLayout());
62:                 String portNum = null;
63:                 if (inAnApplet)
64:                         portNum = getParameter("port");
65:                 port = PORTNUM;
66:                 if (portNum != null)
67:                         port = Integer.parseInt(portNum);
68:
69:                 // The GUI
70:                 ta = new TextArea(14, 80);
71:                 ta.setEditable(false);          // readonly
72:                 ta.setFont(new Font("Monospaced", Font.PLAIN, 11));
73:                 cp.add(BorderLayout.NORTH, ta);
74:
75:                 Panel p = new Panel();
76:
77:                 // The login button
78:                 p.add(lib = new Button("Login"));
79:                 lib.setEnabled(true);
80:                 lib.requestFocus();
81:                 lib.addActionListener(new ActionListener() {
82:                         public void actionPerformed(ActionEvent e) {
83:                                 login();
84:                                 lib.setEnabled(false);
85:                                 lob.setEnabled(true);
86:                                 tf.requestFocus();      // set keyboard focus in right place!
87:                         }
88:                 });
```

```
   89:
   90:                 // The logout button
   91:                 p.add(lob = new Button("Logout"));
   92:                 lob.setEnabled(false);
   93:                 lob.addActionListener(new ActionListener() {
   94:                         public void actionPerformed(ActionEvent e) {
   95:                                 logout();
   96:                                 lib.setEnabled(true);
   97:                                 lob.setEnabled(false);
   98:                                 lib.requestFocus();
   99:                         }
  100:                 });
  101:
  102:                 p.add(new Label("Message here:"));
  103:                 tf = new TextField(40);
  104:                 tf.addActionListener(new ActionListener() {
  105:                         public void actionPerformed(ActionEvent e) {
  106:                                 if (loggedIn) {
  107:                                         pw.println(Chat.CMD_BCAST+tf.getText());
  108:                                         tf.setText("");
  109:                                 }
  110:                         }
  111:                 });
  112:                 p.add(tf);
  113:
  114:                 cp.add(BorderLayout.SOUTH, p);
  115:
  116:         cp.addWindowListener(new WindowAdapter() {
  117:                         public void windowClosing(WindowEvent e) {
  118:                                 // If we do setVisible and dispose, then the Close completes
  119:                                 ChatRoom.this.cp.setVisible(false);
  120:                                 ChatRoom.this.cp.dispose();
  121:                                 logout();
  122:                         }
  123:                 });
  124:                 cp.pack();
  125:                 // After packing the Frame, centre it on the screen.
  126:                 Dimension us = cp.getSize(),
  127:                         them = Toolkit.getDefaultToolkit().getScreenSize();
  128:                 int newX = (them.width - us.width) / 2;
  129:                 int newY = (them.height- us.height)/ 2;
  130:                 cp.setLocation(newX, newY);
  131:                 cp.setVisible(true);
  132:                 paintMessage = "Window should now be visible";
```

```
133:                    repaint();
134:            }
135:
136:            protected String serverHost = "localhost";
137:
138:            /** LOG ME IN TO THE CHAT */
139:            public void login() {
140:                    showStatus("In login!");
141:                    if (loggedIn)
142:                            return;
143:                    if (inAnApplet)
144:                            serverHost = getCodeBase().getHost();
145:                    try {
146:                            sock = new Socket(serverHost, port);
147:                            is = new BufferedReader(new InputStreamReader(sock.getInputStream()));
148:                            pw = new PrintWriter(sock.getOutputStream(), true);
149:                    } catch(IOException e) {
150:                            showStatus("Can't get socket to " +
151:                                    serverHost + "/" + port + ": " + e);
152:                            cp.add(new Label("Can't get socket: " + e));
153:                            return;
154:                    }
155:                    showStatus("Got socket");
156:
157:                    // Construct and start the reader: from server to textarea.
158:                    // Make a Thread to avoid lockups.
159:                    new Thread(new Runnable() {
160:                            public void run() {
161:                                    String line;
162:                                    try {
163:                                            while (loggedIn && ((line = is.readLine()) != null))
164:                                                    ta.append(line + "\n");
165:                                    } catch(IOException e) {
166:                                            showStatus("GAA! LOST THE LINK!!");
167:                                            return;
168:                                    }
169:                            }
170:                    }).start();
171:
172:                    // FAKE LOGIN FOR NOW
173:                    pw.println(Chat.CMD_LOGIN + "AppletUser");
174:                    loggedIn = true;
175:            }
176:
```

```
177:         /** Log me out, Scotty, there's no intelligent life here! */
178:         public void logout() {
179:                 if (!loggedIn)
180:                         return;
181:                 loggedIn = false;
182:                 try {
183:                         if (sock != null)
184:                                 sock.close();
185:                 } catch (IOException ign) {
186:                         // so what?
187:                 }
188:         }
189:
190:         // It is deliberate that there is no STOP method - we want to keep
191:         // going even if the user moves the browser to another page.
192:         // Anti-social? Maybe, but you can use the CLOSE button to kill
193:         // the Frame, or you can exit the Browser.
194:
195:         /** Paint paints the small window that appears in the HTML,
196:          * telling the user to look elsewhere!
197:          */
198:         public void paint(Graphics g) {
199:                 Dimension d = getSize();
200:                 int h = d.height;
201:                 int w = d.width;
202:                 g.fillRect(0, 0, w, 0);
203:                 g.setColor(Color.black);
204:                 g.drawString(paintMessage, 10, (h/2)-5);
205:         }
206:
207:
208:         /** a showStatus that works for Applets or non-Applets alike */
209:         public void showStatus(String mesg) {
210:                 if (inAnApplet)
211:                         super.showStatus(mesg);
212:                 System.out.println(mesg);
213:         }
214:
215:         /** A main method to allow the client to be run as an Application */
216:         public static void main(String[] args) {
217:                 ChatRoom room101 = new ChatRoom();
218:                 room101.inAnApplet = false;
219:                 room101.init();
220:                 room101.start();
```

```
221:         }
222: }
```