

```

1 // fo1/10/juno/RemoteConsole.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.io.*;
7 import java.net.*;
8 import java.util.*;
9 import java.text.*;
10
11 /**
12  * A remote console listens on a port for a remote login to
13  * a running Juno system server.
14  *
15  * @version 1.0
16  */
17
18 public class RemoteConsole extends Thread
19 implements OutputInterface, InputInterface
20 {
21     // default just logs connection start and end
22     // change to true to log all i/o
23     private static boolean logall = false;
24
25     private Juno system;
26     private boolean echo;
27     private InterpreterInterface interpreter;
28
29     private Socket clientSocket;
30     private BufferedReader in;
31     private PrintWriter out;
32     private int sessionCount = 0;
33
34     private PrintWriter junolog;
35
36     /**
37      * Construct a remote console to listen for users trying
38      * to connect to Juno.
39      *
40      * Called from Juno main.
41      *
42      * @param system the Juno system setting up this console.
43      * @param echo whether or not input should be echoed.
44      * @param port the port on which to listen for requests.
45      */
46
47     public RemoteConsole( Juno system, boolean echo, int port )
48     {
49         this.echo = echo;
50         Date now = new Date();
51         junolog = openlog(now);
52         log("*** Juno server started " + now + "\n");
53         try {
54             ServerSocket ss = new ServerSocket(port);
55             while (true) {
56                 clientSocket = ss.accept();

```

```

57         new RemoteConsole( system, echo, clientSocket,
58                             junolog, ++sessionCount).start();
59     }
60 }
61 catch (IOException e) {
62     System.out.println("Remote login not supported");
63     System.exit(0);
64 }
65 finally {
66     system.shutdown();
67 }
68
69 /**
70  * Construct a remote console for a single remote user.
71  *
72  * @param system the Juno system to which the user is connecting.
73  * @param echo whether or not input should be echoed.
74  * @param clientSocket the socket for the user's connection
75  * @param junolog track all user i/o
76  * @param sessionCount this session's number
77  */
78
79 public RemoteConsole( Juno system, boolean echo, Socket clientSocket,
80                       PrintWriter junolog, int sessionCount )
81 {
82     this.system = system;
83     this.echo = echo;
84     this.clientSocket = clientSocket;
85     this.junolog = junolog;
86     this.sessionCount = sessionCount;
87 }
88
89 /**
90  * Action when the thread for this session starts.
91  */
92
93 public void run()
94 {
95     log("*** " + sessionCount + ", " +
96         clientSocket.getInetAddress() + ", " +
97         new Date());
98     try {
99         setUpConnection();
100        String s = this.readLine
101            ("Please sign the guest book (name, email): ");
102        this.println("Thanks, " + s);
103        if (!logall) {
104            log("guest book: " + s);
105        }
106        new LoginInterpreter( system, this ).login();
107        clientSocket.close();
108    }
109    catch (IOException e) {
110        log("*** Error " + e);
111    }
112 }

```

```

113     log("*** end session " + sessionCount);
114     }
115     /**
116     * Create the readers and writers for the socket
117     * for this session.
118     */
119     private void setUpConnection()
120     throws IOException
121     {
122         in = new BufferedReader(
123             new InputStreamReader(clientSocket.getInputStream()));
124         out = new PrintWriter(
125             new OutputStreamWriter(clientSocket.getOutputStream()));
126     }
127     // implement the InputInterface
128     /**
129     * Read a line (terminated by a newline) from console socket.
130     */
131     * Log the input line before returning it if required.
132     * @param promptString output string to prompt for input
133     * @return the string (without the newline character)
134     */
135     public String readline( String promptString )
136     {
137         String s = "";
138         this.print(promptString);
139         out.flush();
140         try {
141             s = in.readLine();
142             if (logall) {
143                 log("> " + s);
144             }
145             if (echo) {
146                 out.println(s);
147             }
148             catch (IOException e) {
149                 String msg = "IO error reading from remote console";
150                 System.out.println(msg);
151                 out.println(msg);
152             }
153             return s;
154         }
155     /**
156     * Write a String to console socket.
157     * Log the output if required.
158     * @param str - the string to write
159     */

```

```

169     */
170     public void print( String str )
171     {
172         out.print( str );
173         out.flush();
174         if (logall) {
175             log("< " + str + "\\");
176         }
177     }
178     // implement the OutputInterface
179     /**
180     * Write a String followed by a newline
181     * to console socket.
182     * Log the output if required.
183     * @param str - the string to write
184     */
185     public void println( String str )
186     {
187         out.println( str + '\r' );
188         out.flush();
189         if (logall) {
190             log("< " + str);
191         }
192     }
193     /**
194     * Write a String followed by a newline
195     * to console error output location. That's
196     * just the socket.
197     * @param str - the String to write
198     */
199     public void errPrintln(String str )
200     {
201         println( str );
202     }
203     /**
204     * Query what kind of console this is.
205     * @return false since it's not a GUI.
206     */
207     public boolean isGUI()
208     {
209         return false;
210     }
211     /**
212     */
213     }
214     }
215     }
216     }
217     }
218     }
219     }
220     }
221     }
222     }
223     }
224     }

```

```
225     * Query what kind of console this is.
226     *
227     * @return true since it is remote.
228     */
229
230     public boolean isRemote()
231     {
232         return true;
233     }
234
235     /**
236     * Query what kind of console this is.
237     *
238     * @return true if and only if echoing input.
239     */
240
241     public boolean isEchoInput()
242     {
243         return echo;
244     }
245
246     /**
247     * Log a String.
248     *
249     * @param str the String to log.
250     */
251
252     private void log(String str)
253     {
254         junolog.println(sessionCount + " : " + str);
255         junolog.flush();
256     }
257
258     /**
259     * Open a log for this console.
260     *
261     * @param now the current Date.
262     */
263
264     private PrintWriter openlog(Date now)
265     {
266         PrintWriter out = null;
267         SimpleDateFormat formatter
268             = new SimpleDateFormat ("MMM.dd:hh:mm:ss");
269         String dateString = formatter.format(now);
270         String filename = "log-" + dateString;
271         try { out = new PrintWriter(
272             new BufferedWriter(
273                 new FileWriter(filename)));
274         }
275         catch (Exception e) {
276             out = new PrintWriter(new FileWriter(FileDescriptor.out));
277         }
278         return out;
279     }
280 }
```