```
// joi/10/juno/RemoteConsole.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;

/**
 * A remote console listens on a port for a remote login to
 * a running Juno system server.
 *
 * @version 10
 */

public class RemoteConsole extends Thread
    implements OutputInterface, InputInterface
{
    // default just logs connection start and end
    // change to true to log all i/o
    private static boolean logall = false;

    private Juno system;
    private boolean echo;
    private InterpreterInterface interpreter;

    private Socket clientSocket;
    private BufferedReader in;
    private PrintWriter out;
    private int sessionCount = 0;

    private PrintWriter junoLog;

    /**
     * Construct a remote console to listen for users trying
     * to connect to Juno.
     *
     * Called from Juno main.
     *
     * @param system the Juno system setting up this console.
     * @param echo whether or not input should be echoed.
     * @param port the port on which to listen for requests.
     */

    public RemoteConsole( Juno system, boolean echo, int port )
    {
        this.echo = echo;
        Date now  = new Date();
        junoLog   = openlog(now);
        log("*** Juno server started " + now + "\n");
        try {
            ServerSocket ss = new ServerSocket(port);
            while (true) {
                clientSocket = ss.accept();
```



```
                new RemoteConsole( system, echo, clientSocket,
                                   junoLog, ++sessionCount).start();
            }
        }
        catch (IOException e) {
            System.out.println("Remote login not supported");
            System.exit(0);
        }
        finally {
            system.shutDown();
        }
    }

    /**
     * Construct a remote console for a single remote user.
     *
     * @param system the Juno system to which the user is connecting.
     * @param echo whether or not input should be echoed.
     * @param clientSocket the socket for the user's connection
     * @param junoLog track all user i/o
     * @param sessionCount this session's number
     */

    public RemoteConsole( Juno system, boolean echo, Socket clientSocket,
                          PrintWriter junoLog, int sessionCount )
    {
        this.system = system;
        this.echo = echo;
        this.clientSocket = clientSocket;
        this.junoLog = junoLog;
        this.sessionCount = sessionCount;
    }

    /**
     * Action when the thread for this session starts.
     */

    public void run()
    {
        log("*** " + sessionCount + ", " +
            clientSocket.getInetAddress()  + ", "  +
            new Date());
        try {
            setUpConnection();
            String s = this.readLine
                ("Please sign the guest book (name, email): ");
            this.println("Thanks, " + s);
            if (!logall) {
                log("guest book: " + s);
            }
            new LoginInterpreter( system, this ).CLILogin();
            clientSocket.close();
        }
        catch (IOException e) {
            log("*** Error " + e);
        }
```

```
113          log("*** end session " + sessionCount);
114      }
115
116      /**
117       * Create the readers and writers for the socket
118       * for this session.
119       */
120
121      private void setUpConnection()
122           throws IOException
123      {
124          in = new BufferedReader(
125                  new InputStreamReader(clientSocket.getInputStream()));
126          out = new PrintWriter(
127                  new OutputStreamWriter(clientSocket.getOutputStream()));
128      }
129
130      // implement the InputInterface
131
132     /**
133      * Read a line (terminated by a newline) from console socket.
134      *
135      * Log the input line before returning it if required.
136      *
137      * @param promptString output string to prompt for input
138      * @return the string (without the newline character)
139      */
140
141      public String readLine( String promptString )
142      {
143          String s = "";
144          this.print(promptString);
145          out.flush();
146          try {
147              s = in.readLine();
148              if (logall) {
149                  log("> " + s);
150              }
151              if (echo) {
152                  out.println(s);
153              }
154          }
155          catch (IOException e) {
156              String msg = "IO error reading from remote console";
157              System.out.println(msg);
158              out.println(msg);
159          }
160          return s;
161      }
162
163      /**
164       * Write a String to console socket.
165       *
166       * Log the output if required.
167       *
168       * @param str - the string to write
```



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