```
 1  // joi/10/juno/LoginInterpreter.java
 2  //
 3  //
 4  // Copyright 2003 Ethan Bolker and Bill Campbell
 5
 6  import java.util.*;
 7
 8  /**
 9   * Interpreter for Juno login commands.
10   *
11   * There are so few commands that if-then-else logic is OK.
12   *
13   * @version 10
14   */
15
16  public class LoginInterpreter
17      implements InterpreterInterface
18  {
19      private static final String LOGIN_COMMANDS =
20          "help, register, <username>, exit";
21
22      private Juno system;             // the Juno object
23      private OutputInterface console; // where output goes
24
25      /**
26       * Construct a new LoginInterpreter for interpreting
27       * login commands.
28       *
29       * @param system the system creating this interpreter.
30       * @param console the Terminal used for input and output.
31       */
32
33      public LoginInterpreter( Juno system, OutputInterface console)
34      {
35          this.system  = system;
36          this.console = console;
37      }
38
39      /**
40       * Set the console for this interpreter.  Used by the
41       * creator of this interpreter.
42       *
43       * @param console the Terminal to be used for input and output.
44       */
45
46      public void setConsole( OutputInterface console)
47      {
48          this.console = console;
49      }
50
51      /**
52       * Simulates behavior at login: prompt.
53       */
54
55      public void CLILogin()
56      {
```



```
57          welcome();
58          boolean moreWork = true;
59          while( moreWork ) {
60              moreWork = interpret(((InputInterface)console).
61                                   readLine( "Juno login: " ) );
62          }
63      }
64
65
66      /**
67       * Parse user's command line and dispatch appropriate
68       * semantic action.
69       *
70       * @param  inputLine the User's instructions.
71       * @return true except for "exit" command
72       *         or null inputLine.
73       */
74
75      public boolean interpret( String inputLine )
76      {
77          if (inputLine == null) {
78              return false;
79          }
80          StringTokenizer st =
81              new StringTokenizer( inputLine );
82          if (st.countTokens() == 0) {
83              return true; // skip blank line
84          }
85          String visitor = st.nextToken();
86          if (visitor.equals( "exit" )) {
87              return false;
88          }
89          if (visitor.equals( "register" )) {
90              register( st );
91          }
92          else if (visitor.equals( "help" )) {
93              help();
94          }
95          else {
96              String password;
97              try {
98                  if (console.isGUI()) {
99                      password = st.nextToken();
100                 }
101                 else {;
102                     password = readPassword( "password: " );
103                 }
104                 User user = system.lookupUser(visitor);
105                 user.matchPassword( password );
106                 new Shell( system, user, console );
107             }
108             catch (Exception e) {
109
110                 // NullPointerException if no such user,
111                 // JunoException if password fails to match -
112                 // message to user doesn't give away which.
```

```
113                 // The sysadmin would probably want a log
114                 // that did keep track.
115                 //
116                 // Other exceptions should be caught in shell()
117
118                 console.println("sorry");
119             }
120         }
121         return true;
122     }
123
124     // Register a new user, giving him or her a login name and a
125     // home directory on the system.
126     //
127     // StringTokenizer argument contains the new user's login name
128     // followed by full real name.
129
130     private void register( StringTokenizer line )
131     {
132         String username = "";
133         String password = "";
134         String realname = "";
135         try {
136             username = line.nextToken();
137             password = line.nextToken();
138             realname = line.nextToken("").trim();
139         }
140         catch (NoSuchElementException e) {
141         }
142
143         if (username.equals("") || password.equals("")
144             || realname.equals("") ) {
145             console.println(
146                   "please supply username, password, real name");
147             return;
148         }
149         User user = system.lookupUser(username);
150
151         if (user != null) {  // user already exists
152             console.println("sorry");
153             return;
154         }
155         if (badPassword( password )) {
156             console.println("password too easy to guess");
157             return;
158         }
159         Directory home = new Directory( username, null,
160                                         system.getUserHomes() );
161
162         user = system.createUser( username, home, password, realname );
163         home.setOwner( user );
164     }
165
166     // test to see if password is unacceptable:
167     //   fewer than 6 characters
168     //   contains only alphabetic characters
```



```
169
170     private boolean badPassword( String pwd )
171     {
172         if (pwd.length() < 6) {
173             return true;
174         }
175         int nonAlphaCount = 0;
176         for (int i=0; i < pwd.length(); i++) {
177             if (!Character.isLetter(pwd.charAt(i))) {
178                 nonAlphaCount++;
179             }
180         }
181         return (nonAlphaCount == 0);
182     }
183
184     // Used for reading the user's password in CLI.
185
186     private String readPassword( String prompt )
187     {
188         String line =
189             ((InputInterface) console).readLine( prompt );
190         StringTokenizer st = new StringTokenizer( line );
191         try {
192             return st.nextToken();
193         }
194         catch ( NoSuchElementException e )  {}
195         return ""; // keeps compiler happy
196     }
197
198
199     // Display a short welcoming message, and remind users of
200     // available commands.
201
202     private void welcome()
203     {
204         console.println( "Welcome to " + system.getHostName() +
205                          " running "   + system.getOS() +
206                          " version "   + system.getVersion() );
207         help();
208     }
209
210     // Remind user of available commands.
211
212     private void help()
213     {
214         console.println( LOGIN_COMMANDS );
215         console.println("");
216     }
217 }
```