

```

1 // foj/7/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 7
14 */
15
16 public class LoginInterpreter
17 {
18     private static final String LOGIN_COMMANDS =
19         "help, register, <username>, exit";
20
21     private Juno    system; // the Juno object
22     private Terminal console; // for i/o
23
24     /**
25      * Construct a new LoginInterpreter for interpreting
26      * login commands.
27      *
28      * @param system the system creating this interpreter.
29      * @param console the Terminal used for input and output.
30      */
31
32     public LoginInterpreter( Juno system, Terminal console )
33     {
34         this.system = system;
35         this.console = console;
36     }
37
38     /**
39      * Set the console for this interpreter.  Used by the
40      * creator of this interpreter.
41      *
42      * @param console the Terminal to be used for input and output.
43      */
44
45     public void setConsole( Terminal console )
46     {
47         this.console = console;
48     }
49
50     /**
51      * Simulates behavior at login: prompt.
52      * CLI stands for "Command Line Interface".
53      */
54     public void CLILogin()
55     {
56

```

```

57         welcome();
58         boolean moreWork = true;
59         while( moreWork ) {
60             moreWork = interpret( console.readLine( "Juno login: " ) );
61         }
62     }
63
64     // Parse user's command line and dispatch appropriate
65     // semantic action.
66     //
67     // return true unless "exit" command or null inputline.
68
69     private boolean interpret( String inputline )
70     {
71         if (inputline == null) return false;
72         StringTokenizer st =
73             new StringTokenizer( inputline );
74         if (st.countTokens() == 0) {
75             return true; // skip blank line
76         }
77         String visitor = st.nextToken();
78         if (visitor.equals( "exit" )) {
79             return false;
80         }
81         if (visitor.equals( "register" )) {
82             register( st );
83         }
84         else if (visitor.equals( "help" )) {
85             help();
86         }
87         else {
88             User user = system.lookupUser( visitor );
89             new Shell( system, user, console );
90         }
91         return true;
92     }
93
94     // Register a new user, giving him or her a login name and a
95     // home directory on the system.
96     //
97     // StringTokenizer argument contains the new user's login name
98     // followed by full real name.
99
100     private void register( StringTokenizer st )
101     {
102         String userName = st.nextToken();
103         String realName = st.nextToken().trim();
104         Directory home = new Directory( userName, null,
105             system.getUserHomes() );
106         User user = system.createUser( userName, home, realName );
107         home.setOwner( user );
108     }
109
110     // Display a short welcoming message, and remind users of
111     // available commands.
112

```

```
113 private void welcome()
114 {
115     console.println( "Welcome to " + system.getHostByName() +
116                     " running " + system.getOS() +
117                     " version " + system.getVersion() );
118     help();
119 }
120
121 // Remind user of available commands.
122 private void help()
123 {
124     console.println( LOGIN_COMMANDS );
125     console.println("");
126 }
127
128 }
```