

```

1 // foj/10/Juno/Shell.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.util.*;
7
8 /**
9  * Models a shell (command interpreter)
10 *
11 * The Shell knows the (Juno) system it's working in,
12 * the user who started it,
13 * and the console to which to send output.
14 *
15 * It keeps track of the the current working directory ( . ) .
16 *
17 * @version 1.0
18 */
19
20 public class Shell
21 implements InterpreterInterface
22 {
23     private Juno system; // The operating system object
24     private User user; // The user logged in
25     private OutputInterface console; // The console for this shell
26     private Directory dot; // The current working directory
27
28     /**
29      * Construct a login shell for the given user and console.
30      *
31      * @param system a reference to the Juno system.
32      * @param user the User logging in.
33      * @param console a Terminal for input and output.
34      */
35
36     Shell( Juno system, User user, OutputInterface console )
37     {
38         this.system = system;
39         this.user = user;
40         this.console = console;
41         dot = user.getHome(); // default current directory
42
43         if (!console.isGUI()) {
44             this.console = console;
45             CLIShell();
46         }
47         else
48             this.console =
49                 new GUIShellConsole("Juno shell for " + user,
50                                     this, console.isEchoInput());
51     }
52
53     // Run the command line interpreter
54     private void CLIShell()
55     {
56

```

```

57     boolean moreWork = true;
58     while(moreWork) {
59         moreWork = interpret( ((InputInterface) console).
60                               readline( getPrompt() ) );
61     }
62     console.println("goodbye");
63 }
64
65 /**
66  * Interpret a String.
67  *
68  * Syntax
69  * <pre>
70  * shellcommand command-arguments
71  * </pre>
72  *
73  * @param inputLine the String to interpret.
74  * @return true unless shell command is logout.
75  */
76
77     public boolean interpret( String inputLine )
78     {
79         StringTokenizer st = stripComments(inputLine);
80         if (st.countTokens() == 0) { // skip blank line
81             return true;
82         }
83         String commandName = st.nextToken();
84         ShellCommand commandObject =
85             system.getCommandTable().lookup( commandName );
86         if (commandObject == null ) {
87             console.errPrintln( "Unknown command: " + commandName );
88             return true;
89         }
90         try {
91             commandObject.doIt( st, this );
92         }
93         catch (ExitShellException e) {
94             return false;
95         }
96         catch (BadShellCommandException e) {
97             console.errPrintln( "Usage: " + commandName + " " +
98                                 e.getCommand().getArgString() );
99         }
100        catch (JunoException e) {
101            console.errPrintln( e.getMessage() );
102        }
103        catch (Exception e) {
104            console.errPrintln( "you should never get here" );
105            console.errPrintln( e.toString() );
106        }
107        return true;
108    }
109
110    // Strip characters from '#' to end of line, create and
111    // return a StringTokenizer for what's left.
112

```

```

113 private StringTokenizer stripComments( String line )
114 {
115     int commentIndex = line.indexOf('#');
116     if (commentIndex >= 0) {
117         line = line.substring(0,commentIndex);
118     }
119     return new StringTokenizer(line);
120 }
121
122 /**
123  * The prompt for the CLI.
124  */
125 * @return the prompt string.
126 */
127
128 public String getPrompt()
129 {
130     return system.getHostname() + ":" + getDot().getPathName() + "> ";
131 }
132
133 /**
134  * The User associated with this shell.
135  */
136 * @return the user.
137 */
138
139 public User getUser()
140 {
141     return user;
142 }
143
144 /**
145  * The current working directory for this shell.
146  */
147 * @return the current working directory.
148 */
149
150 public Directory getDot()
151 {
152     return dot;
153 }
154
155 /**
156  * Set the current working directory for this Shell.
157  */
158 * @param dot the new working directory.
159 */
160
161 public void setDot(Directory dot)
162 {
163     this.dot = dot;
164 }
165
166 /**
167  * The console associated with this Shell.
168

```

```

169     *
170     * @return the console.
171     */
172     public OutputInterface getConsole()
173     {
174         return console;
175     }
176 }
177
178 /**
179  * The Juno object associated with this Shell.
180  */
181 * @return the Juno instance that created this Shell.
182 */
183
184 public Juno getSystem()
185 {
186     return system;
187 }
188 }

```