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

import java.util.*;

/**
 * Models a shell (command interpreter)
 *
 * The Shell knows the (Juno) system it's working in,
 * the User who started it,
 * and the console to which to send output.
 *
 * It keeps track of the the current working directory (.) .
 *
 * @version 10
 */

public class Shell
    implements InterpreterInterface
{
    private Juno system;             // The operating system object
    private User user;               // The user logged in
    private OutputInterface console; // The console for this shell
    private Directory dot;           // The current working directory

    /**
     * Construct a login shell for the given user and console.
     *
     * @param system a reference to the Juno system.
     * @param user the User logging in.
     * @param console a Terminal for input and output.
     */

    Shell( Juno system, User user, OutputInterface console )
    {
        this.system  = system;
        this.user    = user;
        this.console = console;
        dot          = user.getHome(); // default current directory

        if (!console.isGUI()) {
            this.console = console;
            CLIShell();
        }
        else
            this.console =
                new GUIShellConsole("Juno shell for " + user,
                                    this, console.isEchoInput());
    }

    // Run the command line interpreter

    private void CLIShell()
    {
```



```
        boolean moreWork = true;
        while(moreWork) {
            moreWork = interpret( ((InputInterface) console).
                                  readLine( getPrompt() ) );
        }
        console.println("goodbye");
    }

    /**
     * Interpret a String.
     *
     * Syntax
     * <pre>
     *     shellcommand command-arguments
     * </pre>
     *
     * @param inputLine the String to interpret.
     * @return true unless shell command is logout.
     */

    public boolean interpret( String inputLine )
    {
        StringTokenizer st = stripComments(inputLine);
        if (st.countTokens() == 0) {           // skip blank line
            return true;
        }
        String commandName = st.nextToken();
        ShellCommand commandObject =
            system.getCommandTable().lookup( commandName );
        if (commandObject == null ) {
            console.errPrintln( "Unknown command: " + commandName );
            return true;
        }
        try {
            commandObject.doIt( st, this );
        }
        catch (ExitShellException e) {
            return false;
        }
        catch (BadShellCommandException e) {
            console.errPrintln( "Usage: " + commandName + " " +
                                e.getCommand().getArgString() );
        }
        catch (JunoException e) {
            console.errPrintln( e.getMessage() );
        }
        catch (Exception e) {
            console.errPrintln( "you should never get here" );
            console.errPrintln( e.toString() );
        }
        return true;
    }

    // Strip characters from '#' to end of line, create and
    // return a StringTokenizer for what's left.

```

```
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      /**
168       * The console associated with this Shell.
```



```
169       *
170       * @return the console.
171       */
172
173      public OutputInterface getConsole()
174      {
175          return console;
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  }
```