

```

1 // fo1/10/juno/GUIShellConsole.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import javax.swing.*;
7 import java.awt.*;
8 import java.awt.event.*;
9 import java.util.*;
10
11 /**
12  * The GUI to the Juno system Shell.
13  */
14
15 public class GUIShellConsole extends JFrame
16 implements OutputInterface
17 {
18     private static final int FIELDWIDTH = 50;
19     private static final int FIELDHEIGHT = 10;
20
21     // the components on the window
22
23     private JLabel promptLabel = new JLabel();
24     private JTextField commandLine = new JTextField( FIELDWIDTH );
25     private JButton doIt = new JButton( "Do It" );
26     private JButton logout = new JButton( "Logout" );
27     private JTextArea stdout =
28         new JTextArea( FIELDHEIGHT, FIELDWIDTH );
29     private JTextArea stderr =
30         new JTextArea( FIELDHEIGHT/2, FIELDWIDTH );
31
32     private Shell sh; // for interpreting shell commands
33     private WindowCloser closer; // for logging out.
34
35     private boolean echoInput;
36
37     /**
38      * Construct a GUI console for a shell.
39      *
40      * @param title the title to display in the frame.
41      * @param sh the shell to interpret commands.
42      * @param echoInput is input to be echoed?
43      */
44
45     public GUIShellConsole( String title,
46                             Shell sh,
47                             boolean echoInput )
48     {
49         this.sh = sh;
50         this.echoInput = echoInput;
51
52         setTitle( title );
53         setPrompt( sh.getPrompt() );
54
55         // set up console's look and feel
56

```

```

57         JPanel outerPanel = new JPanel();
58         outerPanel.setLayout( new BorderLayout() );
59
60         Box box = Box.createVerticalBox();
61
62         JPanel commandPanel = new JPanel();
63         commandPanel.setLayout( new BorderLayout() );
64         commandPanel.add( promptLabel, BorderLayout.NORTH );
65         commandPanel.add( commandLine, BorderLayout.CENTER );
66         box.add( commandPanel );
67         box.add( Box.createVerticalStrut( 10 ) );
68
69         Box buttons = Box.createHorizontalBox();
70         buttons.add( Box.createGlue() );
71         buttons.add( doIt );
72         buttons.add( Box.createGlue() );
73         buttons.add( logout );
74         buttons.add( Box.createGlue() );
75         box.add( buttons );
76         box.add( Box.createVerticalStrut( 10 ) );
77
78         JPanel stdoutPanel = new JPanel();
79         stdoutPanel.setLayout( new BorderLayout() );
80         stdoutPanel.add( new JLabel( "Standard output:" ),
81                         BorderLayout.NORTH );
82
83         stdoutPanel.add( new JScrollPane( stdout ),
84                         BorderLayout.CENTER );
85
86         box.add( stdoutPanel );
87         box.add( Box.createVerticalStrut( 10 ) );
88         stdout.setEditable( false );
89
90         JPanel stderrPanel = new JPanel();
91         stderrPanel.setLayout( new BorderLayout() );
92         stderrPanel.add( new JLabel( "Error output:" ),
93                         BorderLayout.NORTH );
94         stderrPanel.add( new JScrollPane( stderr ),
95                         BorderLayout.CENTER );
96         box.add( stderrPanel );
97         box.add( Box.createVerticalStrut( 10 ) );
98         stderr.setEditable( false );
99
100        outerPanel.add( box, BorderLayout.CENTER );
101        this.getContentPane().add( outerPanel, BorderLayout.CENTER );
102
103        // Install menus and tool bar.
104
105        JMenuBar menuBar = new JMenuBar();
106        JMenu commandMenu = new JMenu( "Command" );
107        JMenu helpMenu = new JMenu( "Help" );
108
109        JToolBar toolBar = new JToolBar();
110
111        // Create menu items and tool buttons for each shell command
112

```

```

113 ShellCommandTable table = sh.getSystem().getCommandTable();
114 String [] commandNames = table.getCommandNames();
115 for ( int i = 0; i < commandNames.length; i++ ) {
116
117     String commandName = commandNames[i];
118     ShellCommand command =
119         table.lookup( commandName );
120
121     CommandMenuAction commandAction =
122         new CommandMenuAction( commandName,
123             command.getArgString() );
124
125     HelpMenuAction helpAction =
126         new HelpMenuAction( commandName,
127             command.getArgString(),
128             command.getHelpString() );
129
130     JMenuItem item1 = commandMenu.add( commandAction );
131     JMenuItem item2 = helpMenu.add( helpAction );
132     JButton button = toolbar.add( commandAction );
133     JButton button.setTooltipText( command.getHelpString() );
134
135 }
136
137 this.setMenuBar( menuBar );
138 this.getContentPane().add( toolbar,
139     BorderLayout.NORTH );
140 menuBar.add( commandMenu );
141 menuBar.add( helpMenu );
142 pack();
143 show();
144
145 // add listener to the Do It button
146
147 doIt.addActionListener( new Interpreter() );
148
149 // add listener to the Logout button and window closer
150
151 closeMe = new WindowCloser( this );
152 logout.addActionListener( closeMe );
153 this.addWindowListener( closeMe );
154
155 }
156
157 // Set the GUI prompt
158 private void setPrompt( String prompt )
159 {
160     this.promptLabel.setText( prompt );
161 }
162
163 // Implementing the OutputInterface.
164 // Everything goes to the single message area.
165 public void println( String str )
166 {
167     stdout.append( str + "\n" );
168

```

```

169     }
170 }
171 public void errprintln( String str )
172 {
173     stderr.append( str + "\n" );
174 }
175
176 public boolean isGUI()
177 {
178     return true;
179 }
180
181 public boolean isRemote()
182 {
183     return false;
184 }
185
186 public boolean isEchoInput()
187 {
188     return echoInput;
189 }
190
191 // An inner class for the semantics when the user submits
192 // a ShellCommand for execution.
193
194 private class Interpreter
195     implements ActionListener
196 {
197     public void actionPerformed( ActionEvent e )
198     {
199         String str = commandLine.getText();
200         stdout.append( sh.getPrompt() + str + '\n' );
201         if ( sh.interpret( str ) ) {
202             setPrompt( sh.getPrompt() );
203         }
204         else {
205             closeMe.actionPerformed( null );
206         }
207     }
208 }
209
210 private class CommandMenuAction extends AbstractAction
211 {
212     private String argString;
213     private String helpString;
214
215     public CommandMenuAction( String text, String argString )
216     {
217         super( text );
218         this.argString = argString;
219     }
220
221     public void actionPerformed( ActionEvent e )
222     {
223         commandLine.setText( getValue( Action.NAME ) +
224

```

```
225     }
226   }
227
228   private class HelpMenuAction extends AbstractAction
229   {
230     private String argString;
231     private String helpString;
232
233     public HelpMenuAction( String text, String argString,
234                           String helpString )
235     {
236       super( text );
237       this. argString = argString;
238       this.helpString = helpString;
239     }
240
241     public void actionPerformed((ActionEvent e )
242     {
243       stdout.append( getValue( Action.NAME ) + " : " +
244                     helpString );
245     }
246   }
247
248   // A WindowCloser instance handles close events generated
249   // by the underlying window system with its windowClosing
250   // method, and close events from buttons or other user
251   // components with its actionPerformed method.
252   //
253   // The action is to logout and dispose of this window.
254
255   private static class WindowCloser extends WindowAdapter
256   implements ActionListener
257   {
258     Frame myFrame;
259
260     public WindowCloser( Frame frame ) {
261       myFrame = frame;
262     }
263
264     public void windowClosing (WindowEvent e)
265     {
266       this.actionPerformed( null );
267     }
268
269     public void actionPerformed(ActionEvent e)
270     {
271       myFrame.dispose();
272     }
273   }
274 }
```