```java
import java.awt.Cursor;

/**
 * View class.
 *
 * @author Bruce W. Weide
 * @author Paolo Bucci
 */
@SuppressWarnings("serial")
public final class AppendUndoView1 extends JFrame implements AppendUndoView {

    /**
     * Controller object.
     */
    private AppendUndoController controller;

    /**
     * GUI widgets that need to be in scope in actionPerformed method, and
     * related constants. (Each should have its own Javadoc comment, but these
     * are elided here to keep the code shorter.)
     */
    private static final int LINES_IN_TEXT_AREAS = 5,
            LINE_LENGTHS_IN_TEXT_AREAS = 20, ROWS_IN_BUTTON_PANEL_GRID = 1,
            COLUMNS_IN_BUTTON_PANEL_GRID = 3, ROWS_IN_THIS_GRID = 3,
            COLUMNS_IN_THIS_GRID = 1;

    /**
     * Text areas.
     */
    private final JTextArea inputText, outputText;

    /**
     * Buttons.
     */
    private final JButton resetButton, copyButton, undoButton;

    /**
     * No-argument constructor.
     */
```

```
48      public AppendUndoView1() {
49          // Create the JFrame being extended
50
51          /*
52           * Call the JFrame (superclass) constructor with a String parameter to
53           * name the window in its title bar
54           */
55          super("Simple GUI Demo With MVC");
56
57          // Set up the GUI widgets ----------------------------------------
58
59          /*
60           * Create widgets
61           */
62          this.inputText = new JTextArea("", LINES_IN_TEXT_AREAS,
63                  LINE_LENGTHS_IN_TEXT_AREAS);
64          this.outputText = new JTextArea("", LINES_IN_TEXT_AREAS,
65                  LINE_LENGTHS_IN_TEXT_AREAS);
66          this.resetButton = new JButton("Reset");
67          this.copyButton = new JButton("Copy Input");
68          this.undoButton = new JButton("Undo");
69          /*
70           * Text areas should wrap lines, and outputText should be read-only
71           */
72          this.inputText.setEditable(true);
73          this.inputText.setLineWrap(true);
74          this.inputText.setWrapStyleWord(true);
75          this.outputText.setEditable(false);
76          this.outputText.setLineWrap(true);
77          this.outputText.setWrapStyleWord(true);
78          /*
79           * Create scroll panes for the text areas in case text is long enough to
80           * require scrolling in one or both dimensions
81           */
82          JScrollPane inputTextScrollPane = new JScrollPane(this.inputText);
83          JScrollPane outputTextScrollPane = new JScrollPane(this.outputText);
84          /*
85           * Create a button panel organized using grid layout
```

```
 86          */
 87         JPanel buttonPanel = new JPanel(new GridLayout(
 88                 ROWS_IN_BUTTON_PANEL_GRID,
    COLUMNS_IN_BUTTON_PANEL_GRID));
 89         /*
 90          * Add the buttons to the button panel, from left to right
    and top to
 91          * bottom
 92          */
 93         buttonPanel.add(this.resetButton);
 94         buttonPanel.add(this.copyButton);
 95         buttonPanel.add(this.undoButton);
 96
 97         /*
 98          * Organize main window using grid layout
 99          */
100         this.setLayout(new GridLayout(ROWS_IN_THIS_GRID,
    COLUMNS_IN_THIS_GRID));
101         /*
102          * Add scroll panes and button panel to main window, from
    left to right
103          * and top to bottom
104          */
105         this.add(inputTextScrollPane);
106         this.add(buttonPanel);
107         this.add(outputTextScrollPane);
108
109         // Set up the observers
    -------------------------------------------
110
111         /*
112          * Register this object as the observer for all GUI events
113          */
114         this.resetButton.addActionListener(this);
115         this.copyButton.addActionListener(this);
116         this.undoButton.addActionListener(this);
117
118         // Start the main application window
    --------------------------------
119
120         /*
121          * Make sure the main window is appropriately sized for
    the widgets in
122          * it, that it exits this program when closed, and that it
```

```
    becomes
123          * visible to the user now
124          */
125         this.pack();
126         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
127         this.setVisible(true);
128     }
129 
130     /**
131      * Register argument as observer/listener of this; this must
    be done first,
132      * before any other methods of this class are called.
133      *
134      * @param controller
135      *            controller to register
136      */
137     @Override
138     public void registerObserver(AppendUndoController controller)
    {
139         this.controller = controller;
140     }
141 
142     /**
143      * Updates input display based on String provided as argument.
144      *
145      * @param input
146      *            new value of input display
147      */
148     @Override
149     public void updateInputDisplay(String input) {
150         this.inputText.setText(input);
151     }
152 
153     /**
154      * Updates output display based on String provided as
    argument.
155      *
156      * @param output
157      *            new value of output display
158      */
159     @Override
160     public void updateOutputDisplay(String output) {
161         this.outputText.setText(output);
162     }
```

```
163
164     @Override
165     public void actionPerformed(ActionEvent event) {
166         /*
167          * Set cursor to indicate computation on-going; this
    matters only if
168          * processing the event might take a noticeable amount of
    time as seen
169          * by the user
170          */
171
    this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
172         /*
173          * Determine which event has occurred that we are being
    notified of by
174          * this callback; in this case, the source of the event
    (i.e, the widget
175          * calling actionPerformed) is all we need because only
    buttons are
176          * involved here, so the event must be a button press; in
    each case,
177          * tell the controller to do whatever is needed to update
    the model and
178          * to refresh the view
179          */
180         Object source = event.getSource();
181         if (source == this.resetButton) {
182             this.controller.processResetEvent();
183         } else if (source == this.copyButton) {
184
    this.controller.processCopyEvent(this.inputText.getText());
185         }
186         /*
187          * Set the cursor back to normal (because we changed it at
    the beginning
188          * of the method body)
189          */
190         this.setCursor(Cursor.getDefaultCursor());
191     }
192
193     @Override
194     public void updateUndoAllowed(boolean allowed) {
195         this.undoButton.setEnabled(allowed);
196     }
```

```
197
198 }
199
```