```java
import java.awt.Cursor;

/**
 * View class.
 *
 * @author Bruce W. Weide
 * @author Paolo Bucci
 * @author Gage Farmer
 */
@SuppressWarnings("serial")
public final class DemoView1 extends JFrame implements DemoView {

    /**
     * Controller object.
     */
    private DemoController 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 = 2, ROWS_IN_THIS_GRID = 3,
            COLUMNS_IN_THIS_GRID = 1;

    /**
     * Storage String Arrays.
     */
    private ArrayList<String> outputLast = new ArrayList<>();
    private ArrayList<String> inputLast = new ArrayList<>();

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

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

    /**
     * No-argument constructor.
     */
    public DemoView1() {
        // Create the JFrame being extended

        /*
         * Call the JFrame (superclass) constructor with a String parameter to
         * name the window in its title bar
         */
        super("Simple GUI Demo With MVC");

        // Set up the GUI widgets ----------------------------------------

        /*
         * Create widgets
```

```
69          */
70         this.inputText = new JTextArea("", LINES_IN_TEXT_AREAS,
71                 LINE_LENGTHS_IN_TEXT_AREAS);
72         this.outputText = new JTextArea("", LINES_IN_TEXT_AREAS,
73                 LINE_LENGTHS_IN_TEXT_AREAS);
74         this.resetButton = new JButton("Reset");
75         this.appendButton = new JButton("Append");
76         this.undoButton = new JButton("Undo");
77         /*
78          * Text areas should wrap lines, and outputText should be read-only
79          */
80         this.inputText.setEditable(true);
81         this.inputText.setLineWrap(true);
82         this.inputText.setWrapStyleWord(true);
83         this.outputText.setEditable(false);
84         this.outputText.setLineWrap(true);
85         this.outputText.setWrapStyleWord(true);
86         /*
87          * Create scroll panes for the text areas in case text is long enough to
88          * require scrolling in one or both dimensions
89          */
90         JScrollPane inputTextScrollPane = new JScrollPane(this.inputText);
91         JScrollPane outputTextScrollPane = new JScrollPane(this.outputText);
92         /*
93          * Create a button panel organized using grid layout
94          */
95         JPanel buttonPanel = new JPanel(new GridLayout(
96                 ROWS_IN_BUTTON_PANEL_GRID, COLUMNS_IN_BUTTON_PANEL_GRID));
97         /*
98          * Add the buttons to the button panel, from left to right and top to
99          * bottom
100         */
101        buttonPanel.add(this.resetButton);
102        buttonPanel.add(this.appendButton);
103        buttonPanel.add(this.undoButton);
104        /*
105         * Organize main window using grid layout
106         */
107        this.setLayout(new GridLayout(ROWS_IN_THIS_GRID, COLUMNS_IN_THIS_GRID));
108        /*
109         * Add scroll panes and button panel to main window, from left to right
110         * and top to bottom
111         */
112        this.add(inputTextScrollPane);
113        this.add(buttonPanel);
114        this.add(outputTextScrollPane);
115
116        // Set up the observers ------------------------------------------
117
118        /*
119         * Register this object as the observer for all GUI events
120         */
121        this.resetButton.addActionListener(this);
122        this.appendButton.addActionListener(this);
123        this.undoButton.addActionListener(this);
124
125        // Start the main application window ----------------------------
126
127        /*
```

```
128          * Make sure the main window is appropriately sized for the widgets in
129          * it, that it exits this program when closed, and that it becomes
130          * visible to the user now
131          */
132         this.pack();
133         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
134         this.setVisible(true);
135     }
136 
137     /**
138      * Register argument as observer/listener of this; this must be done first,
139      * before any other methods of this class are called.
140      *
141      * @param controller
142      *            controller to register
143      */
144     @Override
145     public void registerObserver(DemoController controller) {
146         this.controller = controller;
147     }
148 
149     /**
150      * Updates input display based on String provided as argument.
151      *
152      * @param input
153      *            new value of input display
154      */
155     @Override
156     public void updateInputDisplay(String input) {
157         this.inputText.setText(input);
158     }
159 
160     /**
161      * Updates output display based on String provided as argument.
162      *
163      * @param output
164      *            new value of output display
165      */
166     @Override
167     public void updateOutputDisplay(String output) {
168         this.outputText.setText(output);
169     }
170 
171     @Override
172     public void actionPerformed(ActionEvent event) {
173         /*
174          * Set cursor to indicate computation on-going; this matters only if
175          * processing the event might take a noticeable amount of time as seen
176          * by the user
177          */
178         this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
179         /*
180          * Determine which event has occurred that we are being notified of by
181          * this callback; in this case, the source of the event (i.e, the widget
182          * calling actionPerformed) is all we need because only buttons are
183          * involved here, so the event must be a button press; in each case,
184          * tell the controller to do whatever is needed to update the model and
185          * to refresh the view
186          */
```

```
187         Object source = event.getSource();
188         if (source == this.resetButton) {
189             if (!this.outputText.getText().equals("")) {
190                 this.outputLast.add(this.outputText.getText());
191                 this.inputLast.add(this.inputText.getText());
192                 this.controller.processResetEvent();
193             } else {
194                 this.outputLast.add("");
195                 this.inputLast.add(this.inputText.getText());
196                 this.controller.processResetEvent();
197             }
198         } else if (source == this.appendButton) {
199             if (!this.outputText.getText().equals("")) {
200                 this.outputLast.add(this.outputText.getText());
201                 this.inputLast.add(this.inputText.getText());
202                 this.controller.processAppendEvent(this.inputText.getText(),
203                         this.outputText.getText());
204             } else {
205                 this.outputLast.add("");
206                 this.inputLast.add(this.inputText.getText());
207                 this.controller.processAppendEvent(this.inputText.getText(),
208                         this.outputText.getText());
209             }
210         } else if (source == this.undoButton) {
211             if (this.outputLast.size() >= 1) {
212                 String out = this.outputLast.remove(this.outputLast.size() - 1);
213                 String in = this.inputLast.remove(this.inputLast.size() - 1);
214                 this.controller.processUndoEvent(out, in);
215             }
216 
217         }
218         /*
219          * Set the cursor back to normal (because we changed it at the beginning
220          * of the method body)
221          */
222         this.setCursor(Cursor.getDefaultCursor());
223     }
224 
225 }
226 
```