```
1 import java.awt.Cursor;
2 import java.awt.GridLayout;
 3 import java.awt.event.ActionEvent;
4
5 import javax.swing.JButton;
 6 import javax.swing.JFrame;
7 import javax.swing.JPanel;
8 import javax.swing.JScrollPane;
9 import javax.swing.JTextArea;
10
11 /**
12 * View class.
13 *
14 * @author Bruce W. Weide
15 * @author Paolo Bucci
16 */
17 @SuppressWarnings ("serial"
18 public final class AppendUndoView1 extends JFrame implements
  AppendUndoView {
19
20
      /**
21
       * Controller object.
22
23
      private AppendUndoController;
24
25
26
       * GUI widgets that need to be in scope in actionPerformed
  method, and
27
       * related constants. (Each should have its own Javadoc
  comment, but these
28
       * are elided here to keep the code shorter.)
29
30
      private static final int LINES_IN_TEXT_AREAS = 5,
              LINE LENGTHS IN TEXT AREAS = 20,
31
  ROWS_IN_BUTTON PANEL GRID = 1.
32
              COLUMNS IN BUTTON PANEL GRID = 2, ROWS IN THIS GRID =
  3,
33
              COLUMNS IN THIS GRID = 1;
34
35
      /**
36
       * Text areas.
37
       */
38
      private final JTextArea inputText, outputText;
39
```

```
40
      /**
41
       * Buttons.
42
       */
43
      private final JButton appendButton, undoButton;
44
45
      /**
46
       * No-argument constructor.
47
       */
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
           */
          this inputText = new JTextArea("", LINES_IN_TEXT_AREAS,
62
                   LINE LENGTHS_IN_TEXT_AREAS);
63
          this outputText = new JTextArea("", LINES_IN_TEXT_AREAS,
64
65
                  LINE_LENGTHS_IN_TEXT_AREAS);
66
          this appendButton = new JButton("Append");
          this undoButton = new JButton("Undo");
67
68
           * Text areas should wrap lines, and outputText should be
69
  read-only
70
           */
71
          this inputText setEditable(true);
72
          this inputText setLineWrap(true);
73
          this.inputText.setWrapStyleWord(true);
          this outputText setEditable(true);
74
75
          this.outputText.setLineWrap(true);
76
          this outputText setWrapStyleWord(true);
77
78
           * Create scroll panes for the text areas in case text is
  long enough to
79
           * require scrolling in one or both dimensions
```

```
80
            */
81
           JScrollPane inputTextScrollPane = new
   JScrollPane(this inputText)
           JScrollPane outputTextScrollPane = new
82
   JScrollPane(this.outputText);
83
84
           * Create a button panel organized using grid layout
85
86
           JPanel buttonPanel = new JPanel(new GridLayout)
87
                   ROWS IN BUTTON PANEL GRID,
   COLUMNS IN BUTTON PANEL GRID);
88
           /*
89
            * Add the buttons to the button panel, from left to right
   and top to
90
           * bottom
91
           */
92
           buttonPanel.add(this.appendButton);
93
           buttonPanel.add(this.undoButton);
94
95
           * Organize main window using grid layout
96
97
           this setLayout (new GridLayout (ROWS IN THIS GRID,
  COLUMNS IN THIS GRID);
98
           /*
99
            * Add scroll panes and button panel to main window, from
   left to right
100
           * and top to bottom
101
            */
102
           this add(inputTextScrollPane);
103
           this add(buttonPanel);
104
           this add(outputTextScrollPane);
105
106
           // Set up the observers
107
108
           /*
           * Register this object as the observer for all GUI events
109
110
111
           this appendButton addActionListener(this);
112
           this undoButton addActionListener(this);
113
114
           // Start the main application window
115
```

```
116
           /*
            * Make sure the main window is appropriately sized for
117
   the widgets in
118
            * it, that it exits this program when closed, and that it
   becomes
119
           * visible to the user now
120
            */
121
           this pack():
           this setDefaultCloseOperation(JFrame EXIT ON CLOSE);
122
123
           this setVisible(true);
124
125
126
       /**
127
        * Register argument as observer/listener of this; this must
   be done first,
128
        * before any other methods of this class are called.
129
130
        * @param controller
131
                     controller to register
132
        */
133
       @Override
134
       public void registerObserver(AppendUndoController controller)
135
           this controller = controller;
136
137
138
       /**
139
        * Updates input display based on String provided as argument.
140
141
        * @param input
142
        *
                     new value of input display
143
        */
144
       @Override
       public void updateInputDisplay(String input) {
145
146
           this inputText setText(input);
147
148
149
       /**
        * Updates output display based on String provided as
150
   argument.
151
        * @param output
152
153
                     new value of output display
        *
154
        */
```

```
155
       @Override
156
       public void updateOutputDisplay(String output) {
           this outputText setText (output);
157
158
159
       @Override
160
161
       public void updateUndoAllowed(boolean allowed) {
162
           this undoButton setEnabled(allowed):
163
164
165
       @Override
       public void actionPerformed(ActionEvent event) {
166
167
168
            * Set cursor to indicate computation on-going; this
   matters only if
169
            * processing the event might take a noticeable amount of
   time as seen
170
            * by the user
171
            */
172
   this_setCursor(Cursor_getPredefinedCursor(Cursor_WAIT CURSOR));
173
174
            * Determine which event has occurred that we are being
   notified of by
175
            * this callback; in this case, the source of the event
   (i.e, the widget
176
            * calling actionPerformed) is all we need because only
   buttons are
177
            * involved here, so the event must be a button press; in
   each case.
178
            * tell the controller to do whatever is needed to update
   the model and
179
            * to refresh the view
180
            */
181
           Object source = event_getSource();
182
           if (source == this appendButton)
183
               this controller processAppendEvent();
184
           } else if (source == this undoButton)
185
   this.controller.processUndoEvent(this.inputText.getText());
186
187
           /*
            * Set the cursor back to normal (because we changed it at
188
   the beginning
```

AppendUndoView1.java Tuesday, April 19, 2022, 6:14 PM 189 * of the method body) 190 */ 191 this setCursor(Cursor getDefaultCursor()); 192 } 193 194 }

195