```java
import java.awt.Cursor;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

/**
 * View class.
 *
 *
 * @author Jeng Zhuang
 */

public final class AppendUndoView1 extends JFrame implements
AppendUndoView {

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

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

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

    /**
     * No-argument constructor.
     */
    public AppendUndoView1() {
        // Create the JFrame being extended

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

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

        /*
         * Create widgets
         */
        this.inputText = new JTextArea("", 5, 20);
        this.outputText = new JTextArea(5, 20);
        this.resetButton = new JButton("Reset");
        this.appendButton = new JButton("Append");
        this.undoButton = new JButton("Undo");
        /*
         * Text areas should wrap lines, and outputText should be read-only
         */
        this.inputText.setEditable(true);
        this.inputText.setLineWrap(true);
        this.inputText.setWrapStyleWord(true);
        this.outputText.setEditable(false);
        this.outputText.setLineWrap(true);
        this.outputText.setWrapStyleWord(true);
        /*
         * Create scroll panes for the text areas in case text is long enough to
         * require scrolling in one or both dimensions
         */
        JScrollPane inputTextScrollPane = new JScrollPane(this.inputText);
        JScrollPane outputTextScrollPane = new JScrollPane(this.outputText);
        /*
         * Create a button panel organized using grid layout
         */
        JPanel buttonPanel = new JPanel(new GridLayout(1, 3));
        /*
         * Add the buttons to the button panel, from left to right and top to
         * bottom
         */
        buttonPanel.add(this.resetButton);
        buttonPanel.add(this.appendButton);
        buttonPanel.add(this.undoButton);
        /*
         * Organize main window using grid layout
         */
        this.setLayout(new GridLayout(3, 1));
        /*
         * Add scroll panes and button panel to main window, from left to right
         * and top to bottom
         */
        this.add(inputTextScrollPane);
        this.add(buttonPanel);
        this.add(outputTextScrollPane);

        // Set up the observers ----------------------------------------------

        /*
         * Register this object as the observer for all GUI events
         */
        this.resetButton.addActionListener(this);
        this.appendButton.addActionListener(this);
        this.undoButton.addActionListener(this);

        // Start the main application window --------------------------------

        /*
         * Make sure the main window is appropriately sized for the widgets in
         * it, that it exits this program when closed, and that it
```

```java
becomes
         * visible to the user now
         */
        this.pack();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    /**
     * Register argument as observer/listener of this; this must be done first,
     * before any other methods of this class are called.
     *
     * @param controller
     *            controller to register
     */
    @Override
    public void registerObserver(AppendUndoController controller) {
        this.controller = controller;
    }

    /**
     * Updates input display based on String provided as argument.
     *
     * @param input
     *            new value of input display
     */
    @Override
    public void updateInputDisplay(String input) {
        this.inputText.setText(input);
    }

    /**
     * Updates output display based on String provided as argument.
     *
     * @param output
     *            new value of output display
     */
    @Override
    public void updateOutputDisplay(String output) {
        this.outputText.setText(output);
    }

    @Override
    public void updateUndoAllowed(boolean allowed) {
        this.undoButton.setEnabled(allowed);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        /*
         * Set cursor to indicate computation on-going; this matters only if
         * processing the event might take a noticeable amount of time as seen
         * by the user
         */
        this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        /*
         * Determine which event has occurred that we are being notified of by
         * this callback; in this case, the source of the event (i.e, the widget
         * calling actionPerformed) is all we need because only buttons are
         * involved here, so the event must be a button press; in each case,
         * tell the controller to do whatever is needed to update the model and
         * to refresh the view
         */
        Object source = event.getSource();
        if (source == this.resetButton) {
            this.controller.processResetEvent();
        } else if (source == this.appendButton) {
            this.controller.processAppendEvent(this.inputText.getText());
        } else if (source == this.undoButton) {
            this.controller.processUndoEvent();
        }
        /*
         * Set the cursor back to normal (because we changed it at the beginning
         * of the method body)
         */
        this.setCursor(Cursor.getDefaultCursor());
    }

}

```