

Brendan Whitaker

CSE 2221 Homework 22

Professor Bucci

18 April 2017

1. Controller1

```
import components.stack.Stack;

/**
 * Controller class.
 *
 * @author Bruce W. Weide
 * @author Paolo Bucci
 */
public final class AppendUndoController1 implements
AppendUndoController {

    /**
     * Model object.
     */
    private final AppendUndoModel model;

    /**
     * View object.
     */
    private final AppendUndoView view;

    /**
     * Updates view to display model.
     *
     * @param model
     *           the model
     * @param view
     *           the view
     */
    private static void updateViewToMatchModel(AppendUndoModel model,
```

```

        AppendUndoView view) {
    /*
     * Get model info
     */
    String input = model.input();
    Stack<String> output = model.output();
    String outputString = "";
    String helper = "";
    for (String elem : output) {
        helper = outputString;
        outputString = helper + elem;
    }

    /*
     * Update view to reflect changes in model
     */
    view.updateInputDisplay(input);
    view.updateOutputDisplay(outputString);
}

/**
 * Constructor; connects {@code this} to the
 * model and view it coordinates.
 *
 * @param model
 *         model to connect to
 * @param view
 *         view to connect to
 */
public AppendUndoController1(AppendUndoModel model,
AppendUndoView view) {
    this.model = model;
    this.view = view;
    /*
     * Update view to reflect initial value of model
     */
    updateViewToMatchModel(this.model, this.view);
}

/**
 * Processes reset event.
 */
@Override
public void processResetEvent() {
    /*
     * Update model in response to this event

```

```

        */
        this.model.setInput("");
        Stack<String> clearStack = this.model.output();
        clearStack.clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    /**
     * Processes Append event.
     *
     * @param input
     *             value of input text
     */
    @Override
    public void processAppendEvent(String input) {
        /*
         * Update model in response to this event
         */
        Stack<String> out = this.model.output();
        out.push(input);
        this.model.setInput("");

        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processUndoEvent() {
        Stack<String> out = this.model.output();

        String in = out.pop();
        this.model.setInput(in);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }
}

```

2. Model1

```
import components.stack.Stack;
import components.stack.Stack1L;

/**
 * Model class.
 *
 * @author Bruce W. Weide
 * @author Paolo Bucci
 */
public final class AppendUndoModel1 implements AppendUndoModel {

    /**
     * Model variables.
     */
    private String input;
    /**
     * More Model variables.
     */
    private Stack<String> output;

    /**
     * Default constructor.
     */
    public AppendUndoModel1() {
        /**
         * Initialize model; both variables start as empty strings
         */
        Stack<String> nullStack = new Stack1L<String>();
        this.input = "";
        this.output = nullStack;
    }

    @Override
    public void setInput(String input) {
        this.input = input;
    }

    @Override
    public String input() {
        return this.input;
    }

    @Override
    public Stack<String> output() {
```

```

        return this.output;
    }
}

```

3. View1

```

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 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 , 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(" Simple GUI Demo With MVC");

    // Set up the GUI widgets

    /*
     * Create widgets
     */
    this.inputText = new JTextArea("", LINES_IN_TEXT_AREAS,
        LINELENGTHS_IN_TEXT_AREAS);
    this.outputText = new JTextArea("", LINES_IN_TEXT_AREAS,
        LINELENGTHS_IN_TEXT_AREAS);
    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(
    ROWS_IN_BUTTON_PANEL_GRID,
    COLUMNS_IN_BUTTON_PANEL_GRID));
/*
 * 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(ROWS_IN_THIS_GRID,
    COLUMNS_IN_THIS_GRID));
/*
 * 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 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 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());
    }
    /*
     * Set the cursor back to normal (because we
     * changed it at the beginning
     * of the method body)
     */
    this.setCursor(Cursor.getDefaultCursor());
}

/**
 * Updates display of whether undo operation is allowed.
 *
 * @param allowed
 *         true iff undo is allowed
 */
@Override
public void updateUndoAllowed(boolean allowed) {

```

}

}