Meriel Stein

Thomas Harker

Della’s House of Bagels

Lab Writeup

**2. Prepare a Word document that describes how the Bagel House operates**

**Your explanation needs to demonstrate that you understand:**

**i. The purpose of each class**

**ii. Which controls (ie components ie software components that a computer user interacts with through direct manipulation to read or edit information about an application) are utilized and which options have been set;**

**iii. How the layout managers are working: i.e., what is the purpose of each layout decision;**

**iv. Any special techniques that were used inside of a layout and why they were used; (i.e., what was the visual result of these techniques);**

**v. The use of panels and frames;**

**vi. Which components have listeners, and the sequence of steps that occurs in Java when controls with listeners are activated. Please write this document in good Java-speak – in other words, try to use the formal Java / OO vocabulary as much as possible.**

2.

The purpose of ToppingPanel allows for the user to choose from a selection of toppings. The ToppingPanel class is a subclass of JPanel, so the constructor ToppingPanel()also calls the default constructor for its superclass, JPanel. The first line of the constructor sets the layout of the ToppingPanel as a GridLayout of four rows and one column so that, as the components are added, they appear in one column in the order they are added. The graphical appearance of the panel itself is contained within a border that is titled “Toppings” using the nested call of the setBorder(BorderFactory.createTitledBorder(“Toppings”) methods. The toppings themselves are represented by objects of type JCheckBox. The one-parameter constructor is used to instantiate the JCheckBox objects, only giving an identifying string, which means that all these JCheckBox objects will be initially unselected. The type JCheckBox is a good choice for this panel because the user can select anywhere from zero to all the toppings and it will be a valid option. After the JCheckBox components are instantiated inside of the constructor, they are added to the panel. ToppingPanel also contains a method getToppingCost()that calculates the total cost of toppings based on the state of the JCheckBox components - specifically, whether or not they have been selected - and the constant variables storing the prices of the various toppings.

The purpose of the CoffeePanel class is to create a panel which displays the various choices of coffee (or lack thereof) that a customer can make. The CoffeePanel class is a subclass of JPanel, so the constructor CoffeePanel() also calls the default constructor for its superclass, JPanel.The class also provides a method for calculating the cost of the coffee variant chosen. The panel utilizes a GridLayout of four rows and one column so the components appear in one column in the order that they are added. A titled border is also set around the panel, titled “Coffee”. A ButtonGroup of JRadioButtons are used to enable to customer to make their choice of coffee. These JRadioButtons are instantiated in the constructor and then added to the ButtonGroup bg. These JRadioButtons are then added to the panel. The decision to use JRadioButton enforces that only one choice of coffee can be made. The CoffeePanel class also contains a calculate() method that calculates the price of your choice of coffee based on the state of the JRadioButtons and returns the price associated with that particular item.

The BagelPanel (which extends JPanel as well) creates a panel which displays two choices of bagel for customers. The panel uses a GridLayout with 2 rows and one column. These rows correspond with the two bagel choices, whiteBagel and wheatBagel. These are represented by two JRadioButtons which are instantiated in the constructor. The use of JRadioButton only enables one choice of bagel by the customer. The radio button for a white bagel is instantiated using an overloaded constructor for JRadioButton. Its constructor is passed the parameter “true”, causing this JRadioButton to be automatically selected at the start of the application. These buttons are then added to a ButtonGroup bg before being added to the panel. The BagelPanel class also contains a calculate() method that calculates the price of your choice of bagel based on the state of the JRadioButtons and returns the value of the constant variable associated with that particular item.

OrderCalculatorGUI extends JFrame, so its constructor also calls the JFrame constructor, designating a title “Order Calculator”. The default close operation of the OrderCalculatorGUI is the exit of the JFrame. The purpose of this class is to instantiate a GreetingPanel, BagelPanel, ToppingPanel, CoffeePanel, and CostPanel. These panels are then added to their respective sections within the BorderLayout of the frame. The JFrame is then packed (to make sure the frame is the correct size in order to hold the things it contains) and made visible ( using setVisible(true) ). BorderLayout gives the GUI the structure it needs, with a greeting/name of the restaurant at the top and an order total at the bottom, similar to an actual bill. The CENTER, EAST, and WEST regions hold the columnar menu items. The OrderCalculatorGUI class adds a CalcButtonListener to the calcButton located in the CostPanel class. CalcButtonListener is one of the two inner classes in OrderCalculatorGUI (the other being ExitButtonListener). The CalcButtonListener class has an actionPerformed() method which uses its instance variables toppingPanel, coffeePanel, and bagelPanel to determine a total cost based on the state of the various buttons of the GUI. This method also incorporates tax into the customer’s total and displays it in the costPanel. The ExitButtonListener class has an actionPerformed() method which closes the application. When the calculate button and/or exit button have been clicked, these components send an ActionEvent object to their respective listeners. These listeners then call their respective actionPerformed() methods.

GreetingPanel puts a greeting message in a panel and displays it.

GreetingPanel also extends JPanel, so its constructor calls the default constructor of its superclass, JPanel. Its constructor is very simple. A new JLabel is created with the message and then added to the current object. No layout is used in this panel, as it only contains one component and the default FlowLayout will center it horizontally.

CostPanel displays a panel with the total cost. The private inner class ExitButtonListener inside of CostPanel handles the event when the user clicks the exit button. CostPanel extends JPanel, and is therefore its subclass. It has only private variables of type JLabel and JButton. The three private JLabels designate the subtotal, tax and total and their x-alignment is set to RIGHT\_ALIGNMENT, meaning that they are going to be lined up by their last characters within the container to which they’ll be added. When they first appear, there is no dollar sign or dollar amount displayed with them. A Box object is instantiated, which is a lightweight container that uses a BoxLayout object as its layout manager. The border of the Box object is set to empty (a border which takes up space but does no drawing) with parameters of top width of 0, left width of 5, bottom width of 0, right width of 0. Because we called createVerticalBox, this Box will put objects one on top of another as they are added. The calcButton and exitButton are instantiated, and an anonymously instantiated ActionListener is added to exitButton of type ExitButtonListener. Another Box object that arranges components horizontally is instantiated via a static method call, and the two buttons are added to it with using hardcoded dimensions. There are also getters and setters for each of CostPanel’s variables.

**3. In the same document, I want you to critically analyze this application. While it is a good example of listeners and layouts, it is a poor example of OO design. Identify as many design issues as you can. Remember to use the following design principles in your analysis: (See Lesson 03 notes.) The DRY principle The Open/Closed principle The Single Responsibility principle**

DRY

* Multiple instances of code for ExitButtonListener, both inside of OrderCalculatorGUI and CostPanel.

Single Responsibility

* OrderCalculatorGUI has too many responsibilities. It shouldn’t contain code for CalcButtonListener or ExitButtonListener, this code should be contained in the CostPanel class. The responsibility of OrderCalculatorGUI is to set up the GUI by setting the layout and adding the correct panels.
* Putting these actionListener inner classes inside CostPanel class will get rid of the need to call .getExitButton() or a .getCalcButton() method inside them. It also means less extreme method chaining in order to add that listener to the “Calculate” button and when changing the values of the JTextFields in the CostPanel. What’s more, all of these changes effected by CalcButtonListener and ExitButtonListener are enacted upon components supported by CostPanel, making CostPanel a better outer class candidate as the instance variables of CostPanel will be visible to these inner classes.

Open/Closed

* The various methods for calculating cost violate the open/closed principle. For example, if Della’s House of Bagels were to add new toppings the method which returns the cost would need to be edited each time. If instead these toppings were placed in a collection and the cost method looped through that collection to determine cost, the method wouldn’t require constant maintenance.