Milestone 4
Questions

Bardia Parmoun 101143006

Guy Morgenshtern 101151430

When refactoring the StoreView class into a GUI, did you take a composition or an inheritance approach? Why?

We used a composition approach.

Each panel we created functioned a very similar way. Since no additional functionality was needed, a composition approach made more sense than inheriting from a JPanel or JLabel and extending its functionality. Using composition will make it easier for future updates to the interface as none of the components have any specific methods that would prevent them from working with other components. In addition, using composition will make it easier for others to read and understand the GUI since the functionality of each component is kept as is. Our panels also included other swing components like JButtons and JLabels. Both Jbuttons and JPanels can exist on their own and since in our case, a JPanel was constructed out of them, the relationship is "has-a" and not "is-a" which implies composition.

Changelog

Product

- A new final attribute called IMAGEPATH was added to this class to keep track of the address of the image that is allocated to each product. The reason this decision was made was because the image is part of the product and as a result it should be an attribute of it.
- As a result of this addition a new getter method was added to product to obtain the value of the IMAGEPATH.

Inventory

- When initializing the products in the inventory the proper image path for the specific product was added.

StoreManager

- For the method checkout and emptyCart the loops were changed to include iterators instead of going through each item individually. This was done to avoid the concurrent modification error that was caused by removing and accessing arraylist items at the same time from the gui side.

StoreView

- A frame attribute was added to StoreView to keep track of the main frame of the gui that is allocated to each user
- An arraylist of JPanels was added to keep track of all product panels. This arraylist keeps track of the individual panel for each product with its information, picture and related buttons.
- A HashMap called productInfoLabels was added. This hashmap keeps track of the label that is allocated to each individual product which has information about its name, price and stock. This was done so the labels would be accessible throughout the class and can be easily updated by the buttons.
- The cartTotalLabel keeps track of the cart total. This was chosen to be a class attribute as well since it needs to be accessible throughout the class to be updated by the functions such as addtocart, removefromcart, and emptycart.
- A method getAddToCartButton(Product product) was created to return a button to add to the product panel for each product
    - An actionListener was included in order to track when the button was clicked and would add stock of the passed in product to the cart accordingly
- A method getRemoveFromCartButton(Product product) was created to return a button to add to the product panel for each product
    - An actionListener was included in order to track when the button was clicked and would remove stock from the passed in product in the cart accordingly
- getViewCartButton(Icon icon) returns a button with an icon image of a shopping cart that is passed in
    - ActionListener was included to call the displayCart method
- getEmptyCartButton() returns a button that when clicked would remove all the items from a users cart
    - ActionListener included
        - Refreshes all the Products’ JLabels with updated stock
    - An extra dialog box was added asking the user if they were sure they wanted to empty the cart
- getCheckoutButton() returns the checkout button, upon clicking the button it calls displayOrder()
    - ActionListener is included
- The main function is now much shorter since most of the functionality of the program is handled through displayGUI. This function now only initializes the required number of storeViews and calls their respective displayGUI methods.
- displayGUI: This function handles the main user interface of the store.
    - The layout of the store consists of:

        - A header panel that includes the name of the store and the welcome message
        - An items panel which includes all of the products that can be bought from the store with their respective pictures, buttons, and labels
        - A menu panel which includes the three buttons for view cart, checkout, and quit.
        - A footer panel that includes the button empty cart.
    - The items with their buttons are all initialized in the displayItems() method which is called by displayGUI.
    - Different components of the frame are added separately instead of having a singular main panel. This was added for the purpose of enabling the scroll bar. Since scroll bar required access to the contentPane of the frame the other components needed to be added to the frame directly as well to keep their order the same.
- Both addToCartUI and removeFromCartUI are now much shorter since they do not need to deal with the different things such as illegal input names or types. All they need to do is calling the proper function ins the storeManager.
- displayItems():
    - This function is used to create the interface for all the items to be displayed to the user. For each product this function creates a new JPanel of the type GridBayLayout. It adds a label to that panel containing information about the product, it also adds an label with an icon for the product. It also has an extra panel that handles the buttons for that specific product. This function then adds this panel to the arraylist of panels for the product and it adds the product label to its respective hashmap.
- displayCart():
    - This function creates a popup dialog box that displays all the items their amounts the user has in their cart, in text form.
    - The function iterates through the user’s cart and generates a string of the user’s current order
    - The function then calculates the subtotal to display as well
- displayOrder(): Prints a receipt of the customer’s order upon clicking checkout
    - This method was added because it had a different text output than the displayCart
        - It also has a different output as it ends in a dialog box prompting the user if they want to exit the program
    - Displays all products the customer has in their cart, and how many
    - String message of the entire order is displayed in a JOptionPane ConfirmDialogBox
        - Upon clicking yes, the store is exited, the frame’s visibility is set to false and disposed, and the program terminates.

The points of the vulnerability in this program also changed. In case of the batch user interface cases such as invalid input or invalid product name needed to be considered. In this case since the user chooses the product and through the related button so cases like invalid product name or invalid input will not occur. However, since we are loading images there can be instances where the specific image is not located in the folder and it cant properly uploaded and that can cause a error. As a result, proper try catch statements were included to avoid that case.