Palm Beach Aire Grocery

2/19/23

Alex Reburn  
Dept of Computer Science  
COP5339  
areburn2019@fau.edu

Faraji Miller   
Dept of Computer Science  
COP5339  
fmiller2012@fau.edu

# Requirement specification

## Functional Requirements

We will be using Swing for our implementation of the GUI for this project.

### User Browses Products by Selecting Aisle [Alex Reburn](mailto:areburn2019@fau.edu)

The user should be able to view the items available for purchase in the store, all while not having to commit to buying anything. this should allow the user to select an aisle and leave without purchase—perhaps through the use of a “back” button. Within this use case, the user will be able to view the quantity of products and add them to their cart.

### User Searches for Item

The user is able to search for an item in the database by selecting the “search for item” option. the user can then look up the item by name, where they are then taken to the information page of the item and can add it to their cart.

### User Checks Out

Once the user is done adding items to their cart, they will select the option to move to checkout, where they will be taken to a cash register program that will sum up the total amount in their cart and request that the user give the right amount (using System.in for simulation). If the money goes over, change is given and the program terminates. If the transaction fails, then the user is taken back to the main menu without completing the transaction.

### Manager Views Customers @fmiller2012

If the login credentials match that of the store manager, then the Manager is able to select a view of all the produce that has been purchased recently, and the users that are associated with the purchase. They will be able to view username and quantity of everything sold. However, they will not be able to see the User’s password.

### Manager Stocks Shelves

The manager selects the option to add products to the stock. they will select the aisle and the product ID. they will then enter the quantity of products to stock on the shelf (System.in is used). The amount that they then have to pay their supplier is then taken from their (the stores) account. similar transaction to the one above in Use Case C.

### Manager Views Finances

The manager can select the “view finances” option on their dashboard to display the finances that they have received from items sold and given for items bought with the Store’s account. This will be indefinite, and the Manager can leave at any time by use of the “back” button.

## Use Case Essentials

### User Adds to Cart by Selecting Aisle [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

1. User logs in
2. System checks login credentials
3. GUI displays User Dashboard
4. User selects “Browse Store”
5. GUI prompts User to enter an aisle number.
6. User enters an aisle number.
7. Products from the selected aisle are displayed.
8. User selects an item (5).
9. Item information is displayed.
10. User adds to cart (8).
11. the Item is added to the cart.

### User Adds to Cart by Selecting Aisle - Variation #1: User Cancels

1. Start from Step 7 from use case: *User Adds to Cart by Selecting Aisle*
2. User clicks “Leave” in the Aisle.
3. System routes user back to the User Dashboard.

### User Adds to Cart by Selecting Aisle - Variation #2: User Leaves the Item View

1. Start from Step 9 from use case: *User Adds to Cart by Selecting Aisle*
2. User clicks “Cancel” on the Item View GUI
3. System returns user to the Aisle view
4. Continue on Step 7 in use case: *User Adds to Cart by Selecting Aisle*

### User Adds to Cart by Searching for Item

1. User logs in
2. System checks login credentials
3. GUI displays User Dashboard
4. User selects “Search for Item”
5. GUI displays prompt:

please enter the name of the desired item.

1. User enters name of desired item
2. List of items containing name string
3. User selects item
4. Item information is displayed.
5. User adds to cart (7).
6. Item is added to the cart.

### User Adds to Cart by Searching for Item - Variation #1: User Cancels

1. Start from Step 7 from use case: *User Adds to Cart by Searching for Item*
2. User clicks “Leave” in the Search for Item
3. System routes user back to the User Dashboard

### User Adds to Cart by Searching for Item - Variation #2: User Leaves the Item View

1. Start from Step 9 from use case: *User Adds to Cart by Searching for item*
2. User clicks “Cancel” on the Item View GUI
3. System returns user to Search for Item View
4. Continue on Step 7 in use case: *User Add to Cart by Searching for Item*

### User Checks Out

1. User logs in
2. System checks login credentials
3. System displays User Dashboard
4. User selects “Checkout”
5. GUI displays a list of items and their prices, as well as the total.
6. GUI displays prompt:

Please enter your money now.

1. User enters their money amount in the correct GUI textbox.
2. User hits “Complete Purchase”
3. System takes the user back to the DashboardViewUI

### Manager Views Customers @fmilller2012

1. Manager logs in
2. System checks login credential
3. GUI displays Admin Dashboard
4. Manager selects “view transaction history”
5. GUI displays views of users associated with purchases, including username, item purchased, and amount purchased.

### Manager Stocks Shelves

1. Manager logs in
2. System checks login credentials
3. GUI displays Admin Dashboard
4. Manager selects “Stock Shelves”
5. GUI displays daily inventory including option to add products to stock
6. Manager selects aisle,
7. Manager enters product ID
8. Manager selects quantity of products to stock
9. GUI displays total amount to be purchased from the store’s account
10. Manager inputs money.
11. System process payment
12. GUI displays newly updated inventory

### Manager Views Finances

1. Manager logs in
2. System checks login credentials
3. GUI displays Admin Dashboard
4. Manager selects “View Finances”
5. GUI displays Shops finances: including money received and money spent

## Use Case Details

### User Logs into System [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

1. User logs into the System as an Account
2. System searches for the Account within the list of Account objects in System.
3. System displays DashboardViewUI

### User Logs into System - Variation #1: User’s Account is a Manager object.

1. Continue from step 2 of use case: *User Logs into System*
2. System displays ManagerViewUI

### User Adds to Cart by Selecting Aisle

1. Continue from use case: *User Logs into System.*
2. User selects JButton with the label “Browse Store.”
3. System displays AisleViewUI with the list of Aisles from the Store object.
4. User enters an aisle number into JTextField.
5. Store locates the Aisle associated with the value from JTextField.
6. List of Items from the selected Aisle are returned.
7. AisleViewUI updates to display the List of Items.
8. User selects an Item.
9. System displays the current Item as the parameter.
10. User enters the amount of the Item they want to add to the Cart.
11. User selects the JButton labeled “Add to Cart.”
12. Item is added to the Cart belonging to the current Account.

### User Adds to Cart by Selecting Aisle - Variation #1: User Cancels

1. Continue from step 3 of use case: *User Adds to Cart by Selecting Aisle.*
2. User selects the JPanel labeled “Cancel”
3. System displays DashboardViewUI.

### User Adds to Cart by Selecting Aisle - Variation #2: User Leaves the Item View

1. Continue from step 9 of use case: *User Adds to Cart by Selecting Aisle.*
2. User selects the JPanel labeled “Leave Item View.”
3. System displays AisleViewUI.

### User Adds to Cart by Searching for Item

1. Continue from use case: *User Logs into System.*
2. User selects JButton with the label “Search for Item.”
3. System displays SearchViewUI.
4. SearchViewUI displays prompt:

*please enter the name of the desired item.*

1. User enters name of Item.
2. System iterates through the list of Items, and displays each Item that contains the requested string.
3. System displays the current Item as the parameter.
4. User enters the amount of the Item they want to add to the Cart.
5. User selects the JButton labeled “Add to Cart.”
6. Item is added to the Cart belonging to the current Account.

### User Adds to Cart by Searching for Item - Variation #1: User Cancels

1. Continue from step 3 of use case: *User Adds to Cart by Searching for Item.*
2. User selects the JPanel labeled “Cancel”
3. System displays DashboardViewUI.

### User Adds to Cart by Searching for Item - Variation #2: User Leaves the Item View

1. Continue from step 7 of use case: *User Adds to Cart by Searching for Item.*
2. User selects the JPanel labeled “Leave Item View.”
3. System displays AisleViewUI.

### User Checks Out

1. Continue from use case: *User Logs into System.*
2. User selects JButton with the label “Proceed to Checkout.”
3. System displays CheckoutViewUI.
4. CheckoutViewUI displays the list of Items and their prices, as well as the total.
5. CheckoutViewUI displays prompt:

*Please enter your money now.*

1. User enters the money amount in the JTextField.
2. User selects JButton with the label “Complete Purchase.”
3. System checks if the typed amount is greater than the total cost of the Items.
4. System stores Transaction
5. System displays AisleViewUI.

### Manager Views Customers @fmiller2012@fau.edu

1. Continue from use case: *User Logs into System - Variation #1: User’s Account is a Manager.*
2. System searches for the Account within the list of Account objects in System.
3. System displays ManagerViewUI
4. Manager selects JButton with the label “View Transaction History.”
5. System displays TransactionViewUI with Transaction from the Store object.

### Manager Stocks Shelves

1. Continue from use case: *User Logs into System - Variation #1: User’s Account is a Manager.*
2. System searches for the Account within the list of Account objects in System.
3. System displays ManagerViewUI
4. Manager selects JButton with the label “Stock Shelves.”
5. System displays StockViewUI that displays a daily inventory item list including the option to add products to stock.
6. Manager selects JComboBox list of items
7. Manager enters in JTextField for the quantity of products to stock
8. System displays the total amount to be purchased for the Store object.
9. Manager selects JButton with the label “Add”.
10. System displays Transaction occurs
11. System displays newly updated inventory on StockViewUI.

### Manager Stocks Shelves - Variation #1: Cannot Afford Products

1. Continue from step 9 of use case: *Manager Stocks Shelves*
2. System displays Transaction does not occur

### Manager Views Finances

1. Continue from use case: *User Logs into System - Variation #1: User’s Account is a Manager.*
2. System searches for the Account within the list of Account objects in System.
3. System displays ManagerViewUI
4. Manager selects JButton with the label “View Finances.”
5. System displays FinanceViewUI that displays Transaction

# Requirement specification

## CRC Cards [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

| **Account** | |
| --- | --- |
| * stores login information | * Store * Cart |

| **Manager** | |
| --- | --- |
| * Accesses different menu based on credentials * Can still perform the same functions as Account | * Account * Store |

| **LoginViewUI** | |
| --- | --- |
| * login | * Account |

| **Cart** | |
| --- | --- |
| * items are added to the cart. * can remove items from the cart. | * Account |

| **Store** | |
| --- | --- |
| * requests the products by aisle * requests products by search | * Aisle * Product * Transaction |

| **Aisle** | |
| --- | --- |
| * retrieves the products that are in the aisle | * Store * Product |

| **Transaction** | |
| --- | --- |
| * stores money spent and money received * stores transaction name * stores products purchased | * TransactionViewUI * Store |

| **Product** | |
| --- | --- |
| * gets the item info including name, quantity, and price. * adds item to the Cart object | * Aisle * Store * Cart |

| **ProductBundle** | |
| --- | --- |
| * A composite of multiple Product objects. | * Product |

@[fmiller2012@fau.edu](mailto:fmiller2012@fau.edu)

| **DiscountedProduct** | |
| --- | --- |
| * An altered form of a normal Product class. | * Product |

| **DashboardViewUI** | |
| --- | --- |
| * Displays list of options for the user to choose from. * Allows User to select from the present options. | * Store * Account * JPanel * JFrame * JButton * AisleViewUI * SearchViewUI |

| **FinanceViewUI** | |
| --- | --- |
| * Displays total amount of money received and money spent | * Store * Account * JFrame * JPanel * JButton * DashboardViewUI * ManagerViewUI * Transaction |

| **TransactionViewUI** | |
| --- | --- |
| * Displays list of purchases, including username, item purchased and amount purchased | * Store * Account * JFrame * JPanel * JButton * DashboardViewUI * ManagerViewUI |

| **AisleViewUI** | |
| --- | --- |
| * Displays aisle information based on the aisle selected * can select an item to change to ItemViewUI | * Store * Account * JPanel * JButton * JFrame * DashboardViewUI * ItemViewUI |

| **StockViewUI** | |
| --- | --- |
| * Displays a daily inventory item list including the option to add products to stock. * Can press button to add item to the cart * Can press enter in textfield to update quantity | * Store * Account * JPanel * JButton * JFrame * JTextfield * ManagerViewUI |

| **CheckoutViewUI** | |
| --- | --- |
| * Displays current cart * Displays current amount due | * Store * Account * JPanel * JButton * JTextField * JFrame * DashboardViewUI * Cart |

| **SearchViewUI** | |
| --- | --- |
| * Displays the items matching the name requested by the user in the search bar. | * Store * Account * JPanel * JFrame * JButton * DashboardViewUI * ItemViewUI |

## Design Patterns [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

### Decorator

| **Name in Design Pattern** | **Actual Name (Product Prices)** |
| --- | --- |
| Component | Item |
| ConcreteComponent | Product |
| Decorator | DiscountedProduct |
| method() | getPrice() |

### Composite

| **Name in Design Pattern** | **Actual Name (Product Bundles)** |
| --- | --- |
| Primitive | Item |
| Composite | ProductBundles |
| Leaf | Product |
| method() | getPrice() |

### Iterator @fmill019@fau.edu

| **Name in Design Pattern** | **Actual Name (Searchable Items)** |
| --- | --- |
| Aggregate | List |
| ConcreteAggregate | LinkedList<Item> |
| Iterator | ProductIterator |
| ConcreteIterator | An anonymous class that implements ProductIterator interface |
| createIterator() | productIterator() |
| next() | nextProduct() |
| isDone() | isDone() |
| currentItem() | Return value of nextProduct() |

### Singleton

| **Name in Design Pattern** | **Actual Name (Manager View)** |
| --- | --- |
| Instance | Manager |

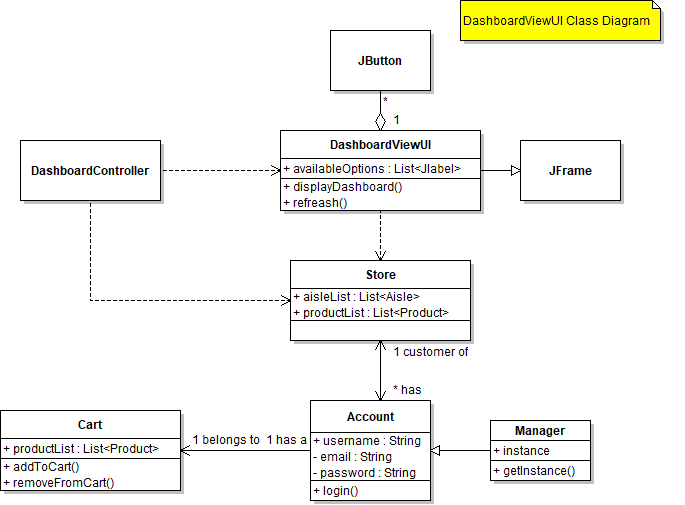
### 

### Observer

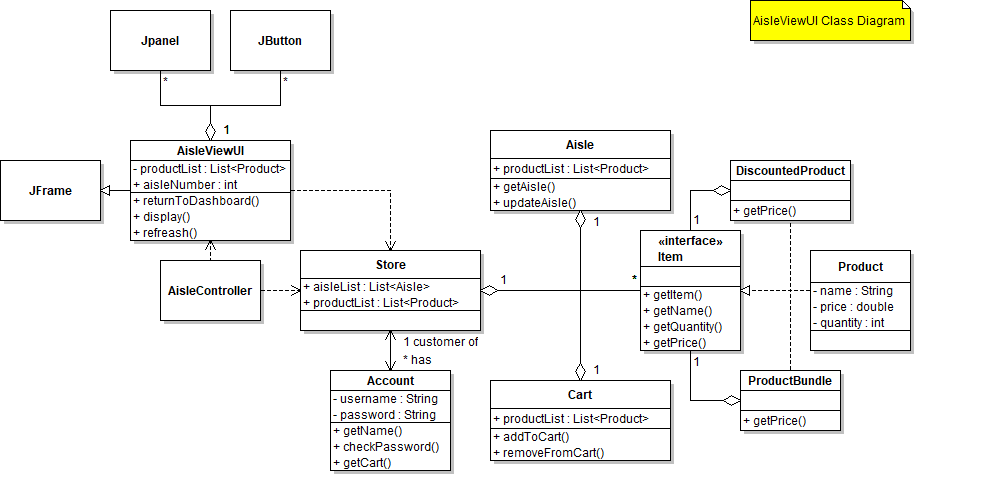
| **Name in Design Pattern** | **Actual Name (Manager View)** |
| --- | --- |
| Subject | JButtonCheckout |
| Observer | ActionListener |
| ConcreteObserver | Product |
| attach() | addActionListener() |
| notify() | actionPerformed() |

## Class Diagrams [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

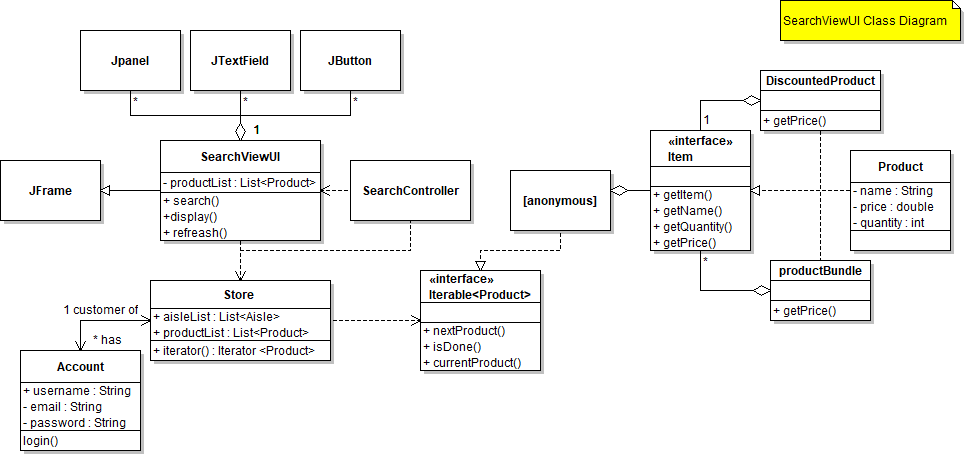
### DashboardViewUI



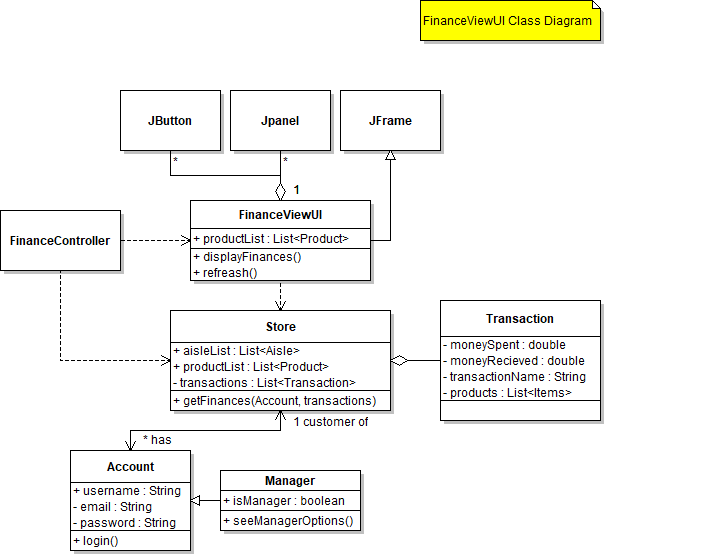
### AisleViewUI



### SearchViewUI

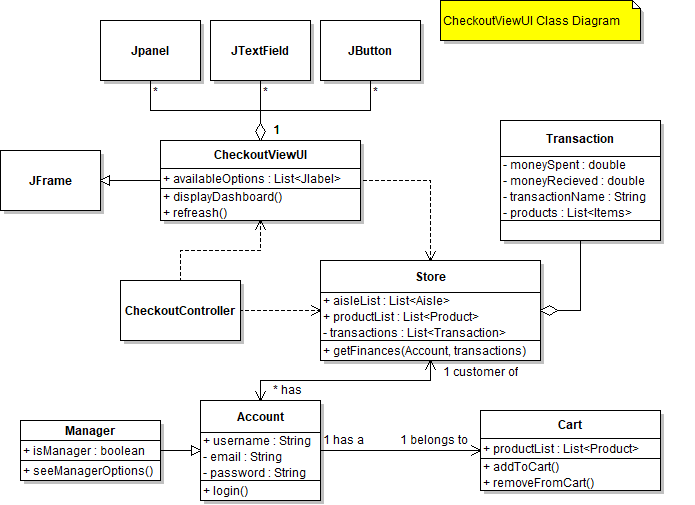


### FinanceViewUI

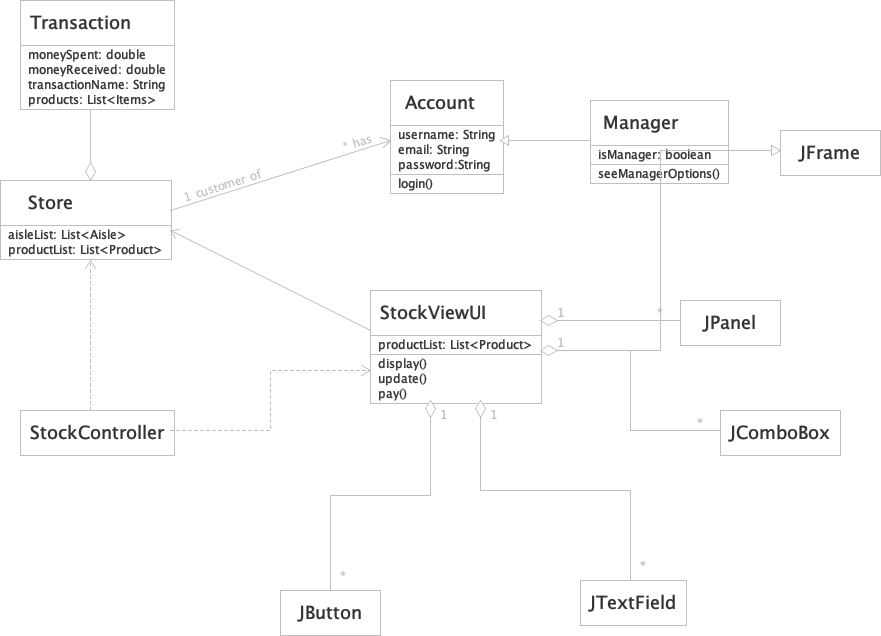


### 

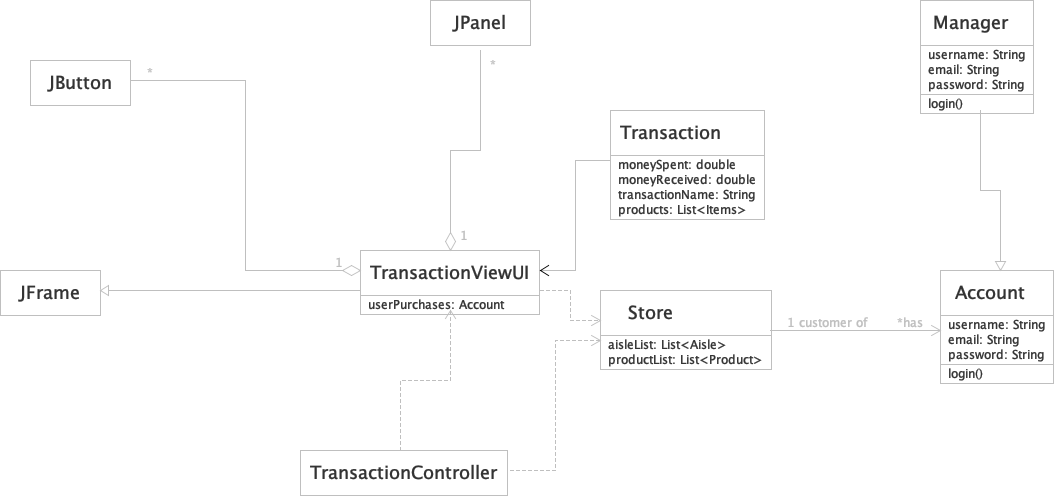
### CheckoutViewUI



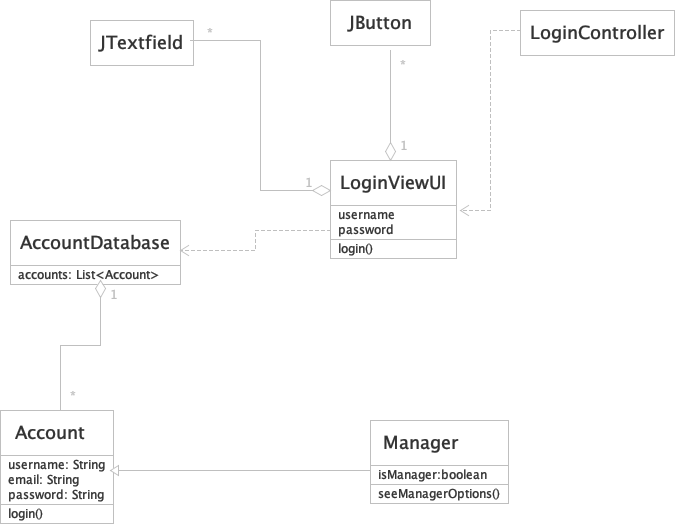
### StockViewUI @fmiller2012@fau.edu



### TransactionViewUI



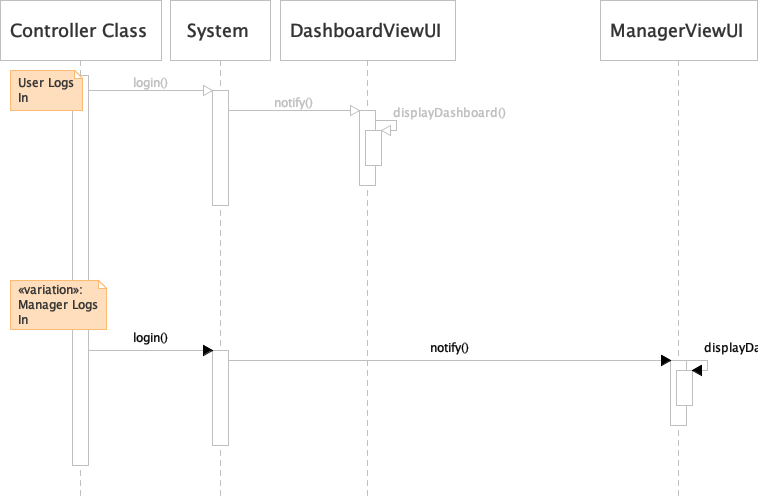
### LoginViewUI



## Sequence Diagrams

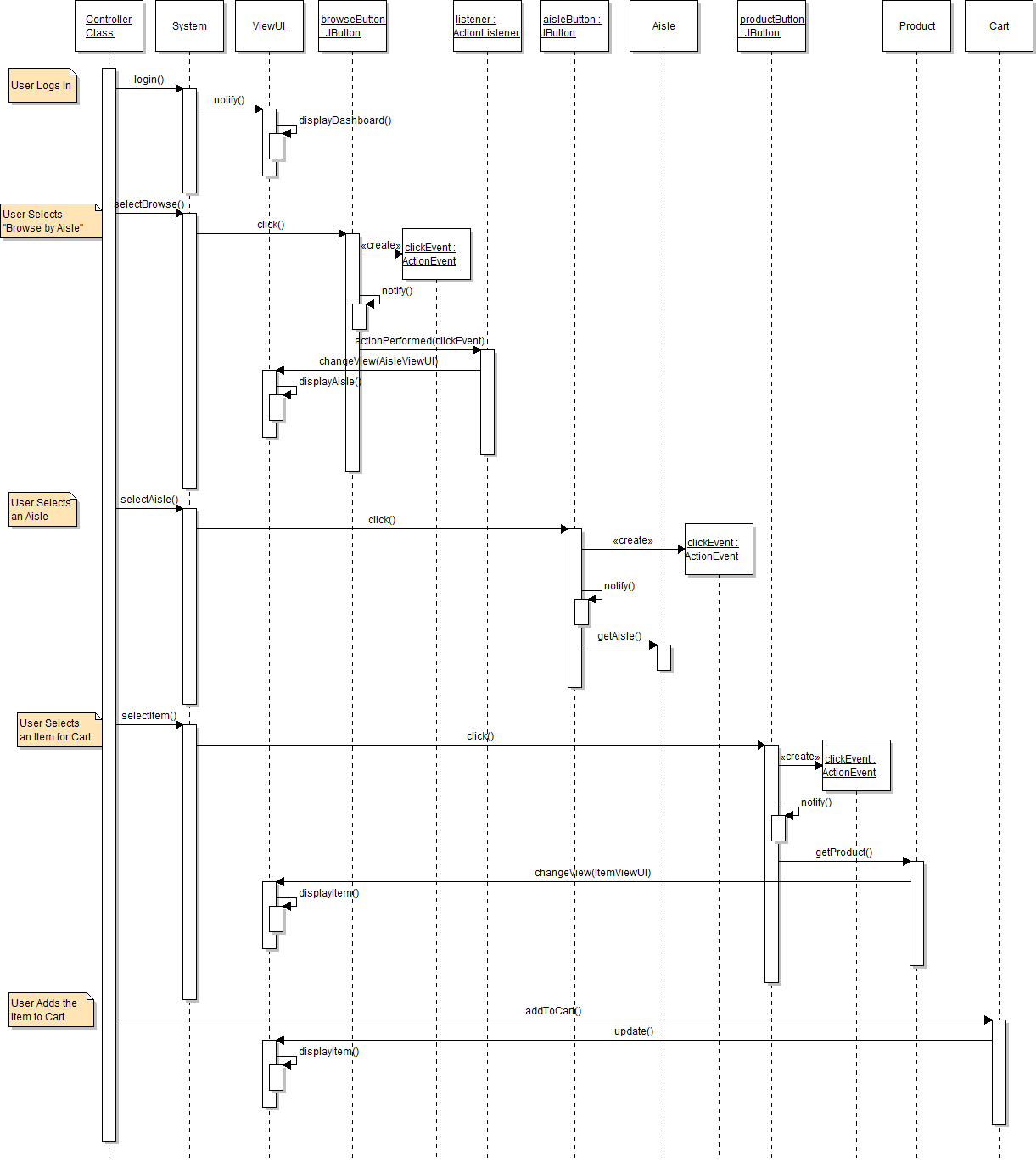
NOTE: the sequence diagrams use ViewUI instead of each individual View (DashboardViewUI, AisleViewUI, etc.) so that the diagram can fit on the graph in a readable manner.

### User/Manager Login @fmiller2012@fau.edu

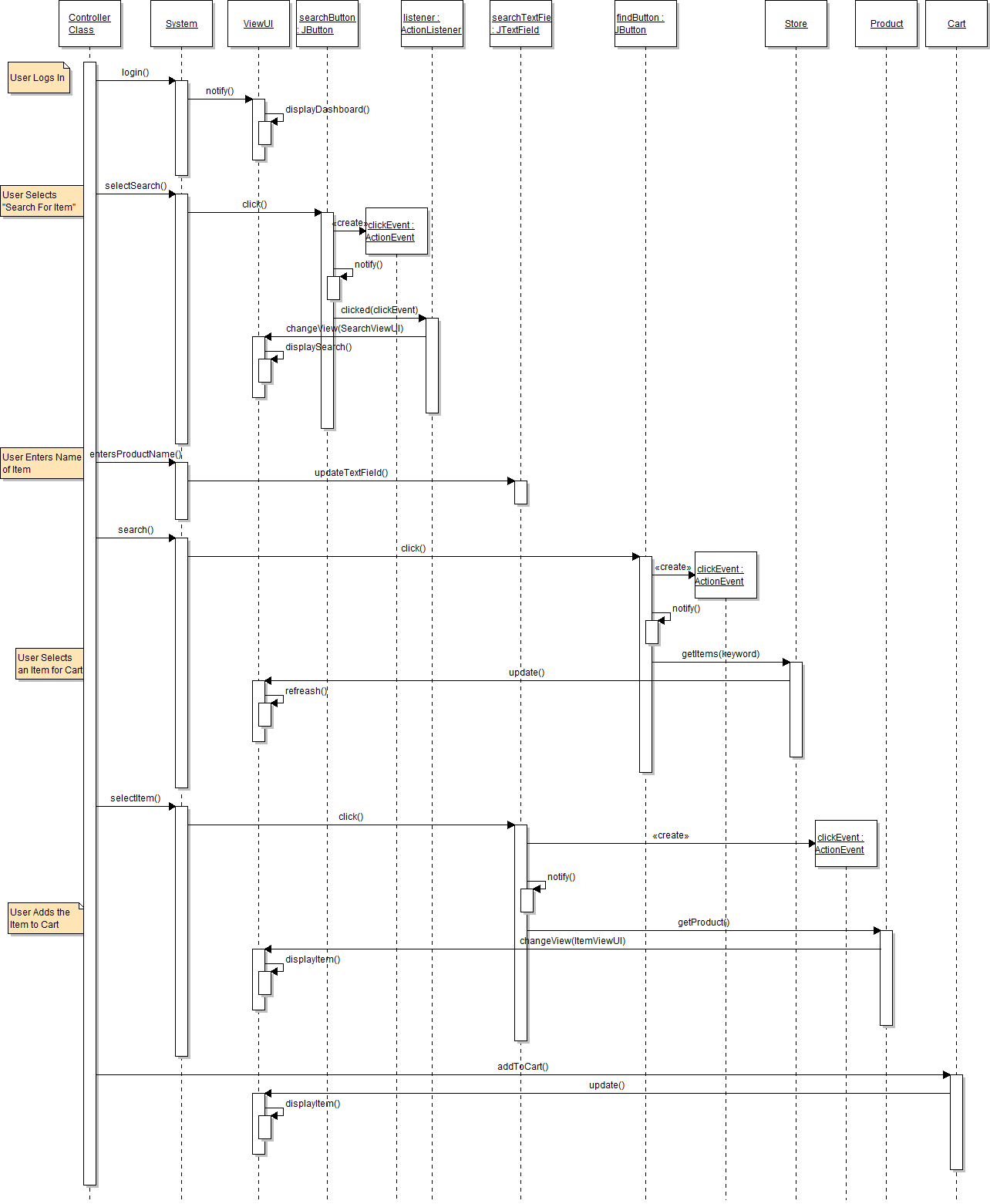


### User Browses Products by Selecting Aisle [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

* + 1. check your phone



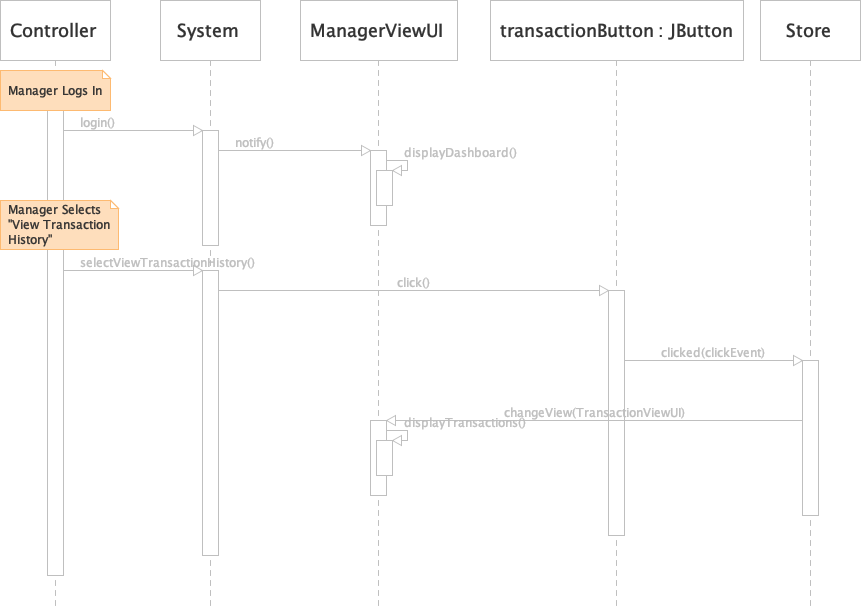
### User Searches for Item



### User Checks Out

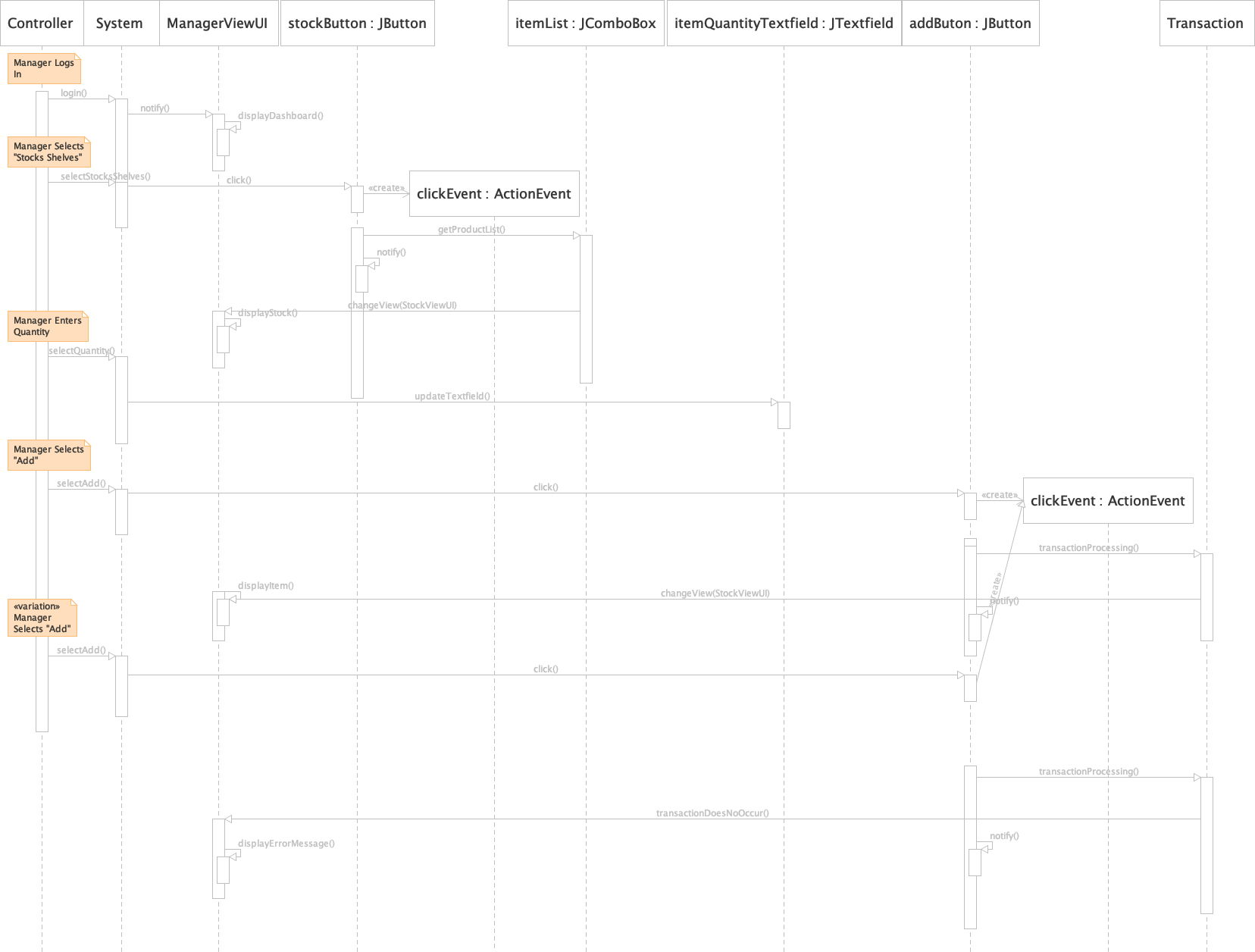
### Manager Views Customers @fmiller2012@fau.edu

NOTE: image faded due to Mac version of Violet Editor



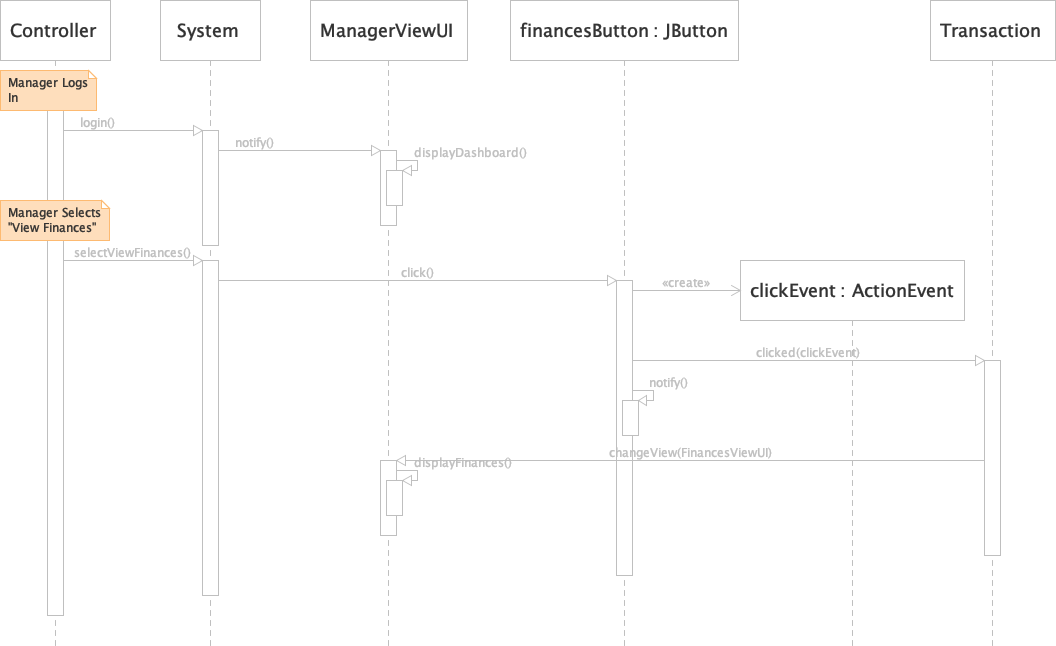
### Manager Stocks Shelves

NOTE: image faded due to Mac version of Violet Editor



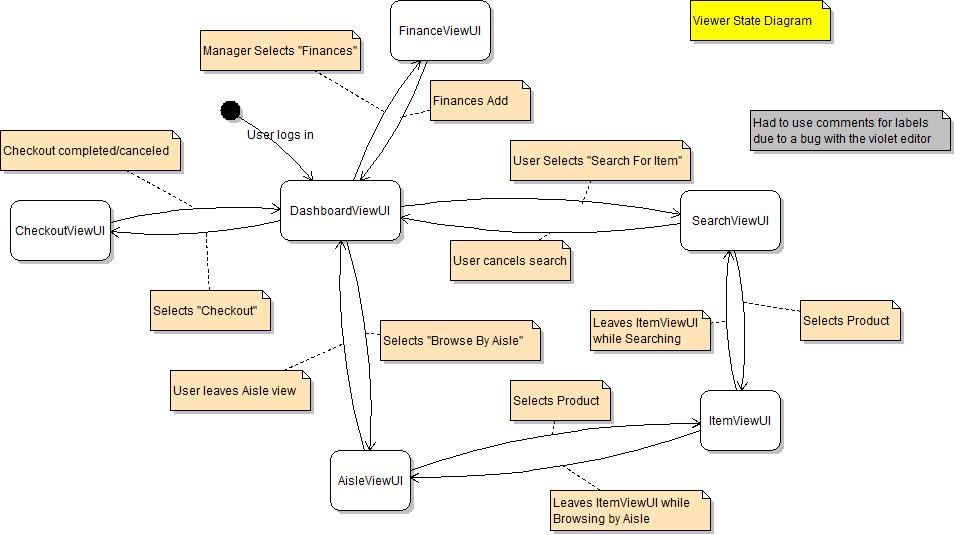
### Manager Views Finances

NOTE: image faded due to Mac version of Violet Editor

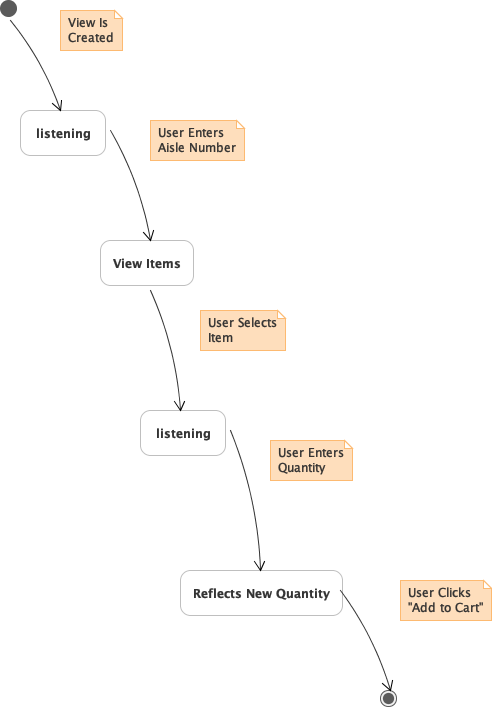


## State Diagrams [areburn2019@fau.edu](mailto:areburn2019@fau.edu)

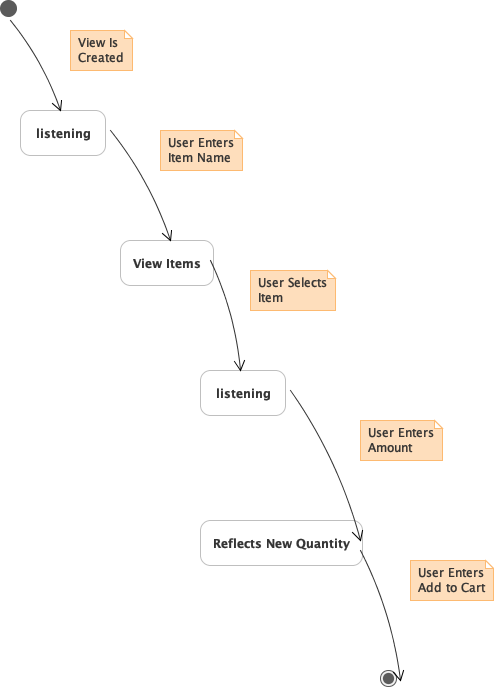
### View State Diagram



### AisleView State Diagram @fmiller2012@fau.edu



### SearchView State Diagram



# Final Report

## Final Analysis

We were unable to include JUnit testing due to time constraints. Below is the code for the assignment. the link to the video is: <https://youtu.be/ocde1UNHYHo>.

## Testing Code

## Models Code

### Account.java

**package** finalProject.models;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Account** {

/\*\*

\* Constructor for the Account object.

\* Initializes the Account object with specified values u, p, and c.

\* @param u the username of the account.

\* @param p the password of the account.

\* @param c the cart object of the account.

\*/

**public** **Account**(String u, String p, Cart c) {

**this**.username = u;

**this**.password = p;

**this**.cart = c;

}

/\*\*

\* Checks if the password is equal to the password in the Database.

\* @param password the current string entered into the password text box.

\* @return whether or not the entered string is the registered password.

\*/

**public** **boolean** **checkPassword**(String password) {

**return** **this**.password.equals(password);

}

/\*\*

\* Returns the current Username.

\* @return username the current username.

\*/

**public** String **getUsername**() {

**return** username;

}

/\*\*

\* returns the Current users cart object.

\* @return cart user's cart.

\*/

**public** Cart **getCart**() {

**return** cart;

}

/\*\*

\* Adds Item to Cart object.

\* @param item item being added to cart.

\*/

**public** **void** **addToCart**(Item item) {

cart.addItem(item);

}

**private** **final** String username;

**private** **final** String password;

**private** Cart cart;

}

### Aisle.java

**package** finalProject.models;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Aisle** {

/\*\*

\* Constructor of the Aisle object.

\* @param aisleNumber The number on the aisle.

\* @param aisleName The name of the aisle.

\*/

**public** **Aisle**(**int** aisleNumber, String aislename) {

**this**.aisleNumber = aisleNumber;

**this**.aisleName = aislename;

**this**.items = **new** ArrayList<Item>();

}

/\*\*

\* Returns the current aisle number.

\* @return aisleNumber the number of the aisle.

\*/

**public** **int** **getAisleNumber**() {

**return** aisleNumber;

}

/\*\*

\* Returns the current aisle name.

\* @return aisleName the name of the aisle.

\*/

**public** String **getAisleName**() {

**return** aisleName;

}

/\*\*

\* Returns the current aisle name.

\* @return aisleName the name of the aisle.

\*/

**public** List<Item> **getItems**() {

**return** items;

}

/\*\*

\* Adds a Product object to the Aisle.

\* @param product the Item being added to the Aisle.

\*/

**public** **void** **addProduct**(Item product) {

items.add(product);

}

**private** **final** **int** aisleNumber;

**private** **final** String aisleName;

**private** **final** List<Item> items;

}

### Cart.java

**package** finalProject.models;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Cart** {

/\*\*

\* Constructor of the Cart object.

\*/

**public** **Cart**() {

**this**.items = **new** ArrayList<>();

}

/\*\*

\* Adds an item to the current Cart object.

\* @param item the item being added to the ArrayList.

\*/

**public** **void** **addItem**(Item item) {

items.add(item);

total += item.getPrice();

}

/\*\*

\* Removes an item to the current Cart object.

\* @param item the item being removed from the ArrayList.

\*/

**public** **void** **removeItem**(Item item) {

items.remove(item);

total -= item.getPrice();

}

/\*\*

\* Returns a list of Items.

\* @return items The list of items in the Cart.

\*/

**public** List<Item> **getItems**() {

**return** items;

}

/\*\*

\* Returns the total of the Cart.

\* @return total The total of the items in the cart.

\*/

**public** **double** **getTotal**() {

**return** total;

}

**private** List<Item> items;

**private** **double** total;

}

### DiscountedProduct.java

**package** finalProject.models;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **DiscountedProduct** **implements** Item {

/\*\*

\* Constructor of the Discounted Product.

\* @param product the product being discounted.

\* @param discount the discount factor of the product price.

\*/

**public** **DiscountedProduct**(Item product, **double** discount) {

**this**.product = product;

**this**.discount = discount;

}

/\*\*

\* Retrieves the String of the Discounted Product.

\* @return the Discounted Product String.

\*/

**@Override**

**public** String **getName**() {

**return** "Discounted " + product.getName() + "(" + **this**.discount + " off)";

}

/\*\*

\* Retrieves the discounted price.

\* @return the price of the discounted price.

\*/

**@Override**

**public** **double** **getPrice**() {

**return** product.getPrice() \* (**1** - discount);

}

/\*\*

\* Retrieves the String of the Discounted Product.

\* @return the Discounted Product String.

\*/

**@Override**

**public** String **toString**() {

**return** **this**.getName();

}

**@Override**

**public** **boolean** **contains**(String text) {

**return** product.getName().toLowerCase().contains(text.toLowerCase());

}

**private** **final** Item product;

**private** **final** **double** discount;

}

### Item.java

**package** finalProject.models;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **interface** **Item** {

**public** **double** **getPrice**();

**public** String **getName**();

**public** **boolean** **contains**(String text);

**@Override**

**public** String **toString**();

}

### Manager.java

**package** finalProject.models;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Manager** **extends** Account {

**private** **static** Manager instance;

**private** **Manager**(String username, String password) {

**super**(username, password, **new** Cart());

}

//static method to get th sigleton instance.

**public** **static** Manager **getInstance**() {

**if**(instance == **null**) {

**synchronized** (Manager.class) {

**if** (instance == **null**) {

instance = **new** Manager("manager", "password");

}

}

}

**return** instance;

}

}

### Product.java

**package** finalProject.models;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Product** **implements** Item {

**public** **Product**(String name, **double** price) {

**this**.name = name;

**this**.price = price;

}

**@Override**

**public** String **getName**() {

**return** name;

}

**@Override**

**public** **double** **getPrice**() {

**return** price;

}

**@Override**

**public** String **toString**() {

**return** **this**.getName();

}

**public** **boolean** **contains**(String text) {

**return** name.toLowerCase().contains(text.toLowerCase());

}

**private** String name;

**private** **double** price;

}

### ProductBundle.java

**package** finalProject.models;

**import** **java.util.ArrayList**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **ProductBundle** **implements** Item {

**private** ArrayList<Item> products;

**public** **ProductBundle**() {

products = **new** ArrayList<>();

}

**public** **void** **addProduct**(Item product) {

products.add(product);

}

**public** **void** **removeProduct**(Item product) {

products.remove(product);

}

**@Override**

**public** String **getName**() {

String bundleName = "Bundle (";

**for** (**int** i = **0**; i < products.size(); i++) {

bundleName += products.get(i).getName();

**if** (i < products.size() - **1**) {

bundleName += ", ";

}

}

bundleName += ")";

**return** bundleName;

}

**@Override**

**public** **double** **getPrice**() {

**double** total = **0**;

**for** (Item product : products) {

total += product.getPrice();

}

**return** total;

}

/\*\*

\* Retrieves the String of the Bundled Product.

\* @return the Discounted Product String.

\*/

**@Override**

**public** String **toString**() {

**return** **this**.getName();

}

**@Override**

**public** **boolean** **contains**(String text) {

**for** (Item product : products) {

**if**(product.getName().toLowerCase().contains(text.toLowerCase())) {

**return** **true**;

}

}

**return** **false**;

}

}

### ProductComboBox.java

**package** finalProject.models;

**import** **java.util.List**;

**import** **javax.swing.JComboBox**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **ProductComboBox** **extends** JComboBox<String>{

**public** **ProductComboBox**(List<Item> l)

{

**super**();

**this**.list = l;

**for**(Item **obj:**list)

{

**this**.addItem(obj.getName());

}

}

List<Item> list;

}

### ProductIterator.java

**package** finalProject.models;

**import** **java.util.Iterator**;

**import** **java.util.List**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **ProductIterator** **implements** Iterator<Item>{

/\*\*

\* Constructor of ProductIterator.

\* @param a the list being iterated.

\*/

**public** **ProductIterator**(List<Item> a) {

**this**.array = a;

}

/\*\*

\* Checks if the Item is the last in the Iterator.

\* @return whether or not the current item is not last.

\*/

**@Override**

**public** **boolean** **hasNext**() {

**return** position < array.size();

}

/\*\*

\* Returns the next Item in the array.

\* @return the next Item in the array.

\*/

**@Override**

**public** Item **next**() {

**if**(!hasNext()) {

**throw** **new** **RuntimeException**("No more elements in array");

}

**return** array.get(position++);

}

**private** **int** position = **0**;

**private** List<Item> array;

}

### Store.java

**package** finalProject.models;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Store** {

**public** **Store**() {

**this**.transactions = **new** ArrayList<>();

**this**.aisles = **new** ArrayList<>();

**this**.createAisles();

}

**public** **void** **addAisle**(Aisle aisle) {

**this**.aisles.add(aisle);

}

**public** List<Aisle> **getAisles**() {

**return** **this**.aisles;

}

**public** Aisle **getAisle**(**int** aisleNumber) {

**for** (Aisle aisle : aisles) {

**if** (aisle.getAisleNumber() == aisleNumber) {

**return** aisle;

}

}

**return** **null**;

}

**public** List<Item> **getItems**(**int** aisleNumber) {

Aisle aisle = getAisle(aisleNumber);

**if** (aisle != **null**) {

**return** aisle.getItems();

}

**return** **null**;

}

**public** **void** **createAisles**() {

Aisle fruitAisle = **new** Aisle(**1**, "Fruit");

fruitAisle.addProduct(**new** Product("Apple", **0.99**));

fruitAisle.addProduct(**new** Product("Banana", **0.69**));

Product discountingItem = **new** Product("Orange", **1.19**);

fruitAisle.addProduct(discountingItem);

fruitAisle.addProduct(**new** DiscountedProduct(discountingItem, **0.2**));

aisles.add(fruitAisle);

Aisle drinksAisle = **new** Aisle(**2**, "Drinks");

drinksAisle.addProduct(**new** Product("Coke", **1.99**));

drinksAisle.addProduct(**new** Product("Sprite", **1.99**));

drinksAisle.addProduct(**new** Product("Pepsi", **1.99**));

aisles.add(drinksAisle);

Aisle meatsAisle = **new** Aisle(**3**, "Meats");

meatsAisle.addProduct(**new** Product("Beef", **5.99**));

meatsAisle.addProduct(**new** Product("Pork", **4.99**));

meatsAisle.addProduct(**new** Product("Chicken", **3.99**));

aisles.add(meatsAisle);

Aisle cheeseAisle = **new** Aisle(**4**, "Cheese");

cheeseAisle.addProduct(**new** Product("Cheddar", **2.99**));

cheeseAisle.addProduct(**new** Product("Mozzarella", **3.99**));

cheeseAisle.addProduct(**new** Product("Swiss", **4.99**));

aisles.add(cheeseAisle);

Aisle candyAisle = **new** Aisle(**5**, "Candy");

candyAisle.addProduct(**new** Product("Gummy Bears", **0.99**));

candyAisle.addProduct(**new** Product("M&M's", **1.29**));

candyAisle.addProduct(**new** Product("Sour Patch Kids", **1.49**));

ProductBundle candyBundle = **new** ProductBundle();

candyBundle.addProduct(**new** Product("Gummy Bears", **0.99**));

candyBundle.addProduct(**new** Product("M&M's", **1.29**));

candyAisle.addProduct(candyBundle);

aisles.add(candyAisle);

products = **new** ArrayList<Item>();

**for**(Aisle a : aisles) {

**for**(Item i : a.getItems()) {

products.add(i);

}

}

}

/\*\*

\* Finds and returns account with credentials matching the input.

\* @param username the string in the username text box.

\* @param password the string in the password text box.

\* @return account the account matching the credentials.

\*/

**public** **static** Account **getAccount**(String username, String password) {

**for** (Account account : accounts) {

**if** (!account.getUsername().equals(username)

|| !account.checkPassword(password)){

} **else** {

**return** account;

}

}

**return** **null**;

}

**public** List<Transaction> **getTransactions**() {

**return** transactions;

}

**public** List<Item> **getItems**() {

**return** products;

}

**private** List<Item> products;

**private** List<Aisle> aisles;

**public** Account currentAccount;

//List of Accounts in the database.

**private** **static** **final** Account[] accounts = {

**new** **Account**("Alex", "password1", **new** Cart()),

**new** **Account**("John", "password2", **new** Cart()),

Manager.getInstance() //Manager is a Singleton.

};

**private** List<Transaction> transactions;

}

### Transaction

**package** finalProject.models;

**import** **java.util.List**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **Transaction** {

/\*\*

\* Constructor for the Transaction Object.

\* @param a

\* @param products

\*/

**public** **Transaction**(Account a, List<Item> products) {

**this**.account = a;

**this**.products = products;

//gets the total from the transaction.

**for** (Item product : products) {

total += product.getPrice();

}

}

/\*\*

\* Retrieves the name of the Account object.

\* @return the username of the Account.

\*/

**public** String **getName**() {

**return** account.getUsername();

}

/\*\*

\* Retrieves the total of the Transaction.

\* @return total the total amount for the transaction.

\*/

**public** **double** **getTotal**() {

**return** total;

}

**private** Account account;

**private** List<Item> products;

**private** **double** total;

}

## Controllers Code

### AisleController.java

**package** finalProject.controllers;

**import** **finalProject.views.DashboardViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.views.AisleViewUI**;

**import** **java.awt.event.ActionEvent**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **AisleController** {

**public** **AisleController**(AisleViewUI view, Store store) {

**this**.view = view;

**this**.store = store;

//adds a button listener for the user selecting an aisle.

view.addAisleButtonListener((ActionEvent e) -> {

//Retrieves aisle information.

view.selectAisle(view.getAisleNumber());

});

//adds a button listener for the user adding an item to their cart.

view.addItemButtonListener((ActionEvent e) -> {

view.addToCart(); //tells the AisleViewUI to add the item to cart.

//Returns the User to a dashboard.

DashboardViewUI dashView = **new** DashboardViewUI(store);

dashView.setVisible(**true**);

view.setVisible(**false**); //Hides the AisleViewUI

});

//adds a button listener for the user cancelling the AisleViewUI.

view.addCancelButtonListener((ActionEvent e) -> {

//Returns the User to a dashboard.

DashboardViewUI dashView = **new** DashboardViewUI(store);

dashView.setVisible(**true**);

view.setVisible(**false**); //Hides the AisleViewUI

});

}

**private** **final** AisleViewUI view;

**private** **final** Store store;

}

### CheckoutController.java

**package** finalProject.controllers;

**import** **finalProject.views.CheckoutViewUI**;

**import** **finalProject.models.Transaction**;

**import** **finalProject.models.Store**;

**import** **finalProject.views.DashboardViewUI**;

**import** **finalProject.models.Item**;

**import** **java.awt.event.ActionEvent**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **javax.swing.JOptionPane**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **CheckoutController** {

/\*\*

\* Constructor that creates the Controller function.

\* @param view the current CheckoutViewUI.

\* @param store the store model.

\*/

**public** **CheckoutController**(CheckoutViewUI view, Store store) {

//Adds a button listener for the user cancelling the CheckoutViewUI.

view.addCancelButtonListener((ActionEvent e) -> {

//Display DashboardViewUI

DashboardViewUI dashView = **new** DashboardViewUI(store);

dashView.setVisible(**true**);

view.setVisible(**false**); //Hides the CheckoutViewUI

});

//Adds a button listener for the user confirming the CheckoutViewUI.

view.addPurchaseButtonListener((ActionEvent e) -> {

**double** cartCost = store.currentAccount.getCart().getTotal();

**double** moneyInput = Double.parseDouble(view.moneyField.getText());

List<Item> items = **new** ArrayList<>(store.currentAccount.getCart().getItems());

**if**(cartCost >= moneyInput) { //if the user does not have enough money.

JOptionPane.showMessageDialog(**null**,

"Insufficient money. Please re-enter currency.");

} **else** { //if the user has enough money.

//Removes items from the Users cart.

**for** (Item product : items) {

store.currentAccount.getCart().removeItem(product);

}

//Adds the transaction to the list stored in the Store object.

Transaction t = **new** Transaction(store.currentAccount, items);

store.getTransactions().add(t);

DashboardViewUI dashView = **new** DashboardViewUI(store);

dashView.setVisible(**true**); //Display DashboardViewUI

view.setVisible(**false**); //Hides the CheckoutViewUI

}

});

}

}

### DashboardController.java

**package** finalProject.controllers;

**import** **finalProject.views.CheckoutViewUI**;

**import** **finalProject.views.DashboardViewUI**;

**import** **finalProject.views.LoginViewUI**;

**import** **finalProject.views.SearchViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.views.AisleViewUI**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **DashboardController** {

/\*\*

\* Constructor that creates the Controller function.

\* @param view the current DashboardViewUI.

\* @param store the store model.

\*/

**public** **DashboardController**(DashboardViewUI view, Store store) {

**this**.view = view;

**this**.store = store;

}

/\*\*

\* Changes view to the AisleViewUI.

\*/

**public** **void** **browseByAisle**() {

AisleViewUI aisleView = **new** AisleViewUI(store);

aisleView.setVisible(**true**); //Display AisleViewUI

view.setVisible(**false**); //Hides the DashboardViewUI

}

/\*\*

\* Changes view to the SearchViewUI.

\*/

**public** **void** **searchForItem**() {

SearchViewUI searchView = **new** SearchViewUI(store);

searchView.setVisible(**true**); //Display SearchViewUI

view.setVisible(**false**); //Hides the DashboardViewUI

}

/\*\*

\* Changes view to the CheckoutViewUI.

\*/

**public** **void** **proceedToCheckout**() {

CheckoutViewUI checkoutView = **new** CheckoutViewUI(store);

checkoutView.setVisible(**true**); //Display CheckoutViewUI

view.setVisible(**false**); //Hides the DashboardViewUI

}

/\*\*

\* Changes view to the LoginViewUI.

\*/

**public** **void** **logout**() {

LoginViewUI checkoutView = **new** LoginViewUI(store);

checkoutView.setVisible(**true**); //Display LoginViewUI

view.setVisible(**false**); //Hides the DashboardViewUI

}

**private** **final** DashboardViewUI view;

**private** **final** Store store;

}

### FinanceController.java

**package** finalProject.controllers;

**import** **finalProject.models.Store**;

**import** **finalProject.views.ManagerViewUI**;

**import** **finalProject.views.FinanceViewUI**;

**import** **java.awt.event.ActionEvent**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **FinanceController** {

**public** **FinanceController**(FinanceViewUI view, Store store) {

//Adds a button listener for the user cancelling the CheckoutViewUI.

view.addLeaveButtonListener((ActionEvent e) -> {

//Display DashboardViewUI

ManagerViewUI dashView = **new** ManagerViewUI(store);

dashView.setVisible(**true**);

view.setVisible(**false**); //Hides the TransactionViewUI

});

}

}

### LoginController.java

**package** finalProject.controllers;

**import** **finalProject.views.DashboardViewUI**;

**import** **finalProject.views.LoginViewUI**;

**import** **finalProject.models.Manager**;

**import** **finalProject.views.ManagerViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.models.Account**;

**import** **javax.swing.JOptionPane**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **LoginController** {

**public** **LoginController**( LoginViewUI view, Store store) {

**this**.view = view;

**this**.store = store;

}

**public** **void** **login**(String username, String password) {

// Attempt to find an account with the given username and password

Account account = store.getAccount(username, password);

**if** (account == **null**) {

// If no account is found, show an error message

JOptionPane.showMessageDialog(**null**, "Invalid username or password. Please try again.");

} **else** {

// If an account is found, check if it's a manager account

**if** (account **instanceof** Manager) {

// If it's a manager account, show the manager view

store.currentAccount = account;

ManagerViewUI managerViewUI = **new** ManagerViewUI(store);

managerViewUI.setVisible(**true**);

view.setVisible(**false**);

} **else** {

store.currentAccount = account;

// If it's a regular user account, show the user dashboard view

DashboardViewUI dashboardViewUI = **new** DashboardViewUI(store);

dashboardViewUI.setVisible(**true**);

view.setVisible(**false**);

}

}

}

**private** LoginViewUI view;

**private** Store store;

}

### ManagerController.java

**package** finalProject.controllers;

**import** **finalProject.views.ManagerViewUI**;

**import** **finalProject.views.StockViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.views.TransactionViewUI**;

**import** **finalProject.views.FinanceViewUI**;

**import** **finalProject.views.LoginViewUI**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **ManagerController** {

**private** **final** ManagerViewUI view;

**private** **final** Store store;

**public** **ManagerController**(ManagerViewUI view, Store store) {

**this**.view = view;

**this**.store = store;

}

**public** **void** **viewTransactionHistory**() {

// Create and show the TransactionViewUI

TransactionViewUI transactionView = **new** TransactionViewUI(store);

transactionView.setVisible(**true**);

// Hide the ManagerViewUI

view.setVisible(**false**);

}

**public** **void** **viewStock**() {

// Create and show the StockViewUI

StockViewUI stockView = **new** StockViewUI(store);

stockView.setVisible(**true**);

// Hide the ManagerViewUI

view.setVisible(**false**);

}

**public** **void** **viewFinances**() {

// Create and show the FinanceViewUI

FinanceViewUI financeView = **new** FinanceViewUI(store);

financeView.setVisible(**true**);

// Hide the ManagerViewUI

view.setVisible(**false**);

}

**public** **void** **logout**() {

LoginViewUI checkoutView = **new** LoginViewUI(store);

checkoutView.setVisible(**true**); //Display LoginViewUI

view.setVisible(**false**); //Hides the DashboardViewUI

}

}

### SearchController.java

**package** finalProject.controllers;

**import** **finalProject.views.SearchViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.models.ProductIterator**;

**import** **finalProject.views.DashboardViewUI**;

**import** **finalProject.models.Item**;

**import** **java.awt.event.ActionEvent**;

**import** **java.util.ArrayList**;

**import** **java.util.Iterator**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **SearchController** {

/\*\*

\* Constructor that creates the Controller function.

\* @param view the current DashboardViewUI.

\* @param store the store model.

\*/

**public** **SearchController**(SearchViewUI view, Store store) {

**this**.view = view;

**this**.store = store;

//Adds a button listener for the user cancelling the SearchViewUI.

view.addCancelButtonListener((ActionEvent e) -> {

DashboardViewUI dashView = **new** DashboardViewUI(store);

dashView.setVisible(**true**); //Display DashboardViewUI

view.setVisible(**false**); //Hides the SearchViewUI

});

//Adds a button listener for the user searching for an Item.

view.addSearchButtonListener((ActionEvent e) -> {

//Creates ArrayList for items matching Product.

ArrayList<Item> matchingProduct = **new** ArrayList<>();

Iterator<Item> iterator = **new** ProductIterator(store.getItems());

**while** (iterator.hasNext()) {

Item item = iterator.next();

**if** (item.contains(view.searchField.getText())) {

matchingProduct.add(item);

}

}

view.displayMatchingProducts(matchingProduct);

});

//adds a button listener for the user adding an item to their cart.

view.addItemButtonListener((ActionEvent e) -> {

Item selectedItem = (Item) view.itemComboBox.getSelectedItem();

**int** quantity = Integer.parseInt(view.quantityField.getText());

**for** (**int** i=**0**; i<quantity; i++) {

store.currentAccount.addToCart(selectedItem);

}

//Returns the User to a dashboard.

DashboardViewUI dashView = **new** DashboardViewUI(store);

dashView.setVisible(**true**);

view.setVisible(**false**); //Hides the AisleViewUI

});

}

**private** **final** SearchViewUI view;

**private** **final** Store store;

}

### StockController.java

**package** finalProject.controllers;

**import** **finalProject.models.Cart**;

**import** **finalProject.models.Item**;

**import** **finalProject.views.ManagerViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.models.Transaction**;

**import** **finalProject.views.StockViewUI**;

**import** **java.awt.event.ActionEvent**;

**import** **java.util.List**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **StockController** {

**public** **StockController**(StockViewUI view, Store store) {

**this**.view = view;

**this**.store = store;

//adds a button listener for the user selecting an aisle.

view.addAisleButtonListener((ActionEvent e) -> {

//Retrieves aisle information.

view.selectAisle(view.getAisleNumber());

});

//adds a button listener for the user adding an item to their cart.

view.addItemButtonListener((ActionEvent e) -> {

Item selectedItem = (Item) view.itemComboBox.getSelectedItem();

**int** quantity = Integer.parseInt(view.quantityField.getText());

**for** (**int** i=**0**; i<quantity; i++) {

store.currentAccount.addToCart(selectedItem);

}

List<Item> currentCart = store.currentAccount.getCart().getItems();

Transaction transaction = **new** Transaction(store.currentAccount, currentCart);

store.getTransactions().add(transaction);

//Returns the User to a dashboard.

ManagerViewUI managerView = **new** ManagerViewUI(store);

managerView.setVisible(**true**);

view.setVisible(**false**); //Hides the AisleViewUI

});

//adds a button listener for the user cancelling the AisleViewUI.

view.addCancelButtonListener((ActionEvent e) -> {

//Returns the User to a dashboard.

ManagerViewUI managerView = **new** ManagerViewUI(store);

managerView.setVisible(**true**);

view.setVisible(**false**); //Hides the AisleViewUI

});

}

**private** **final** StockViewUI view;

**private** **final** Store store;

}

### TransactionController.java

**package** finalProject.controllers;

**import** **finalProject.views.TransactionViewUI**;

**import** **finalProject.models.Store**;

**import** **finalProject.views.ManagerViewUI**;

**import** **java.awt.event.ActionEvent**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **TransactionController** {

**public** **TransactionController**(TransactionViewUI view, Store store) {

//Adds a button listener for the user cancelling the CheckoutViewUI.

view.addLeaveButtonListener((ActionEvent e) -> {

//Display DashboardViewUI

ManagerViewUI dashView = **new** ManagerViewUI(store);

dashView.setVisible(**true**);

view.setVisible(**false**); //Hides the TransactionViewUI

});

}

}

## Views Code

### AisleViewUI.java

**package** finalProject.views;

**import** **finalProject.models.Aisle**;

**import** **finalProject.controllers.AisleController**;

**import** **finalProject.models.Item**;

**import** **finalProject.models.Store**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.Container**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionListener**;

**import** **javax.swing.\***;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **AisleViewUI** **extends** JFrame {

/\*\*

\* Constructor creates a ViewUI for the Aisle.

\* @param store the Model of the MVC.

\*/

**public** **AisleViewUI**(Store store) {

//Setup: Creates a Window for the AisleView.

**super**("Aisle View");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

**this**.store = store; //Conencts the store to the AisleViewUI.

//Setup: a Panel object that will contain each of the objects in frame.

aislePanel = **new** JPanel(**new** GridLayout(**3**, **1**));

aislePanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

//Setup: Creating the J-Objects within the Panel.

cancelButton = **new** JButton("Cancel");

aisleLabel = **new** JLabel("Select an aisle number:");

aisleNumberField = **new** JTextField();

aisleButton = **new** JButton("Select Aisle");

addToCartButton = **new** JButton("Add to Cart");

itemComboBox = **new** JComboBox();

//Create a new AisleController instance to handle button clicks.

AisleController aisleController = **new** AisleController(**this**, store);

//Adds the J-Objects into the Panel.

aislePanel.add(aisleLabel);

aislePanel.add(aisleNumberField);

aislePanel.add(aisleButton);

aislePanel.add(cancelButton);

//Creates the Container.

containerPanel = getContentPane();

containerPanel.add(aislePanel, BorderLayout.NORTH);

//Designs the layout for the ViewUI.

pack();

setTitle("Aisle View");

setSize(**300**, **300**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

/\*\*

\* Adds an action listener to the aisleButton.

\* @param listener the event listener.

\*/

**public** **void** **addAisleButtonListener**(ActionListener listener) {

aisleButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the addToCartButton.

\* @param listener the event listener.

\*/

**public** **void** **addItemButtonListener**(ActionListener listener) {

addToCartButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the cancelButton.

\* @param listener the event listener.

\*/

**public** **void** **addCancelButtonListener**(ActionListener listener) {

cancelButton.addActionListener(listener);

}

/\*\*

\* Displays a different Aisle view that contains an item picker.

\* @param aisleNumber the aisle that is being looked into.

\*/

**public** **void** **selectAisle**(**int** aisleNumber) {

containerPanel.removeAll(); //clears the previous view.

//Setup: a new Panel for selecting the items.

itemPanel = **new** JPanel(**new** GridLayout(**6**, **1**));

itemPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

//Creates an Label for selecting an item.

JLabel itemLabel = **new** JLabel("Select an item:");

Aisle currentAisle = store.getAisle(aisleNumber);

JLabel aisleLabel = **new** JLabel(currentAisle.getAisleName());

//Removes the items from a combo box only if it already has content.

**if** (itemComboBox != **null**){

itemComboBox.removeAllItems();

}

//Adds every product from the Aisle to the JComboBox.

**for** (Item product : currentAisle.getItems()) {

itemComboBox.addItem(product);

}

//Creates a Label and TextBox for quantity.

JLabel quantityLabel = **new** JLabel("Enter the quantity:");

quantityField = **new** JTextField();

//Adds the J-Objects to the Panel.

itemPanel.add(aisleLabel);

itemPanel.add(itemLabel);

itemPanel.add(itemComboBox);

itemPanel.add(quantityLabel);

itemPanel.add(quantityField);

itemPanel.add(addToCartButton);

//Resets the container with the new Panel.

containerPanel.add(itemPanel, BorderLayout.NORTH);

containerPanel.revalidate();

containerPanel.repaint(); //Repaints the container.

}

/\*\*

\* Adds the selected Item to the current users Cart.

\*/

**public** **void** **addToCart**() {

Item selectedItem = (Item) itemComboBox.getSelectedItem();

**int** quantity = Integer.parseInt(quantityField.getText());

**for** (**int** i=**0**; i<quantity; i++) {

store.currentAccount.addToCart(selectedItem);

}

}

/\*\*

\* Get the aisle number entered by the user.

\* @return the number entered into the text field.

\*/

**public** **int** **getAisleNumber**() {

**return** Integer.parseInt(aisleNumberField.getText());

}

**private** **final** Store store;

**private** Container containerPanel;

**private** JPanel aislePanel;

**private** JPanel itemPanel;

**private** **final** JLabel aisleLabel;

**private** JComboBox itemComboBox;

**private** JTextField aisleNumberField;

**private** **final** JButton aisleButton;

**private** **final** JButton cancelButton;

**private** JTextField quantityField;

**private** **final** JButton addToCartButton;

}

### CheckoutViewUI.java

**package** finalProject.views;

**import** **finalProject.controllers.CheckoutController**;

**import** **finalProject.models.Store**;

**import** **finalProject.models.Item**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.Container**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionListener**;

**import** **javax.swing.\***;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **CheckoutViewUI** **extends** JFrame {

/\*\*

\* Constructor for the View of the Checkout functions.

\* @param store the Model of the MVC.

\*/

**public** **CheckoutViewUI**(Store store) {

//Setup: Creates a Window for the CheckoutViewUI.

checkoutPanel = **new** JPanel(**new** GridLayout(**5**, **1**));

checkoutPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

productListPanel = **new** JPanel(**new** GridLayout(**5**, **1**));

//creates a JLabel for all of the Items.

**for** (Item product : store.currentAccount.getCart().getItems()) {

productListPanel.add(**new** JLabel("Item: " + product.getName()

+ ", Price Each: " + product.getPrice()));

}

//Setup: Creating the J-Objects within the Panel.

cancelButton = **new** JButton("Cancel");

totalLabel = **new** JLabel("Total: "

+ store.currentAccount.getCart().getTotal());

moneyField = **new** JTextField();

purchaseButton = **new** JButton("Checkout Items");

//Create a new CheckoutController instance to handle button clicks

CheckoutController checkoutController = **new** CheckoutController(**this**, store);

//Adds the J-Objects into the Panel.

checkoutPanel.add(productListPanel);

checkoutPanel.add(totalLabel);

checkoutPanel.add(moneyField);

checkoutPanel.add(purchaseButton);

checkoutPanel.add(cancelButton);

//Creates the Container.

containerPanel = getContentPane();

containerPanel.add(checkoutPanel, BorderLayout.NORTH);

//Designs the layout for the ViewUI.

pack();

setTitle("Checkout View");

setSize(**500**, **500**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

/\*\*

\* Adds an action listener to the cancelButton.

\* @param listener the event listener.

\*/

**public** **void** **addCancelButtonListener**(ActionListener listener) {

cancelButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the purchaseButton.

\* @param listener the event listener.

\*/

**public** **void** **addPurchaseButtonListener**(ActionListener listener) {

purchaseButton.addActionListener(listener);

}

**private** Container containerPanel;

**private** JPanel checkoutPanel;

**private** JPanel productListPanel;

**private** **final** JButton purchaseButton;

**private** **final** JButton cancelButton;

**public** JTextField moneyField;

**private** **final** JLabel totalLabel;

}

### DashboardViewUI.java

**package** finalProject.views;

**import** **finalProject.models.Store**;

**import** **finalProject.controllers.DashboardController**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.GridLayout**;

**import** **javax.swing.\***;

**import** **java.awt.event.ActionEvent**;

**import** **java.awt.event.ActionListener**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **DashboardViewUI** **extends** JFrame {

/\*\*

\* Constructor for the View of the Checkout functions.

\* @param store the Model of the MVC.

\*/

**public** **DashboardViewUI**(Store store) {

//Setup: Creates a Window for the CheckoutViewUI.

browseByAisleButton = **new** JButton("Browse by Aisle");

searchForItemButton = **new** JButton("Search for Item");

proceedToCheckoutButton = **new** JButton("Proceed to Checkout");

leaveToLoginButton = **new** JButton("Logout");

//Create a new DashboardController instance to handle button clicks

dashboardController = **new** DashboardController(**this**, store);

//Adds the J-Objects into the Panel.

JPanel buttonPanel = **new** JPanel();

buttonPanel.setLayout(**new** GridLayout(**4**, **1**));

buttonPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

buttonPanel.add(browseByAisleButton);

buttonPanel.add(searchForItemButton);

buttonPanel.add(proceedToCheckoutButton);

buttonPanel.add(leaveToLoginButton);

//Sets up the button action listener

browseByAisleButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

// Call the browseByAisle method of the DashboardController to handle the button click

dashboardController.browseByAisle();

}

});

//Sets up the button action listener

searchForItemButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

//Calls the searchForItem method to handle the button click

dashboardController.searchForItem();

}

});

//Sets up the button action listener

proceedToCheckoutButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

//Calls the proceedToCheckout method to handle the button click.

dashboardController.proceedToCheckout();

}

});

leaveToLoginButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

//Calls the proceedToCheckout method to handle the button click.

dashboardController.logout();

}

});

add(buttonPanel, BorderLayout.CENTER);

// Set up the window properties

setTitle("Dashboard");

setSize(**300**, **300**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

**private** **final** JButton browseByAisleButton;

**private** **final** JButton searchForItemButton;

**private** **final** JButton proceedToCheckoutButton;

**private** **final** JButton leaveToLoginButton;

**private** DashboardController dashboardController;

}

### FinanceViewUI

**package** finalProject.views;

**import** **finalProject.controllers.FinanceController**;

**import** **finalProject.models.Manager**;

**import** **finalProject.models.Transaction**;

**import** **finalProject.models.Store**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.Container**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionListener**;

**import** **javax.swing.\***;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **FinanceViewUI** **extends** JFrame {

**public** **FinanceViewUI**(Store store) {

//Setup: Creates a Window for the FinanceViewUI.

historyPanel = **new** JPanel(**new** GridLayout(**5**, **1**));

historyPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

transactionListPanel = **new** JPanel(**new** GridLayout(**5**, **1**));

**double** totalCost = **0**;

//creates a JLabel for all of the Items.

**for** (Transaction t : store.getTransactions()) {

**if**("manager".equals(t.getName())) {

transactionListPanel.add(**new** JLabel("Account: " + t.getName()

+ ", Money: -" + t.getTotal()));

totalCost -= t.getTotal();

} **else** {

transactionListPanel.add(**new** JLabel("Account: " + t.getName()

+ ", Money: " + t.getTotal()));

totalCost += t.getTotal();

}

}

totalLabel = **new** JLabel("Total: " + totalCost);

//Setup: Creating the J-Objects within the Panel.

leaveButton = **new** JButton("Leave");

//Create a new FinanceController instance to handle button clicks

FinanceController financeController = **new** FinanceController(**this**, store);

//Adds the J-Objects into the Panel.

historyPanel.add(transactionListPanel);

historyPanel.add(totalLabel);

historyPanel.add(leaveButton);

//Creates the Container.

containerPanel = getContentPane();

containerPanel.add(historyPanel, BorderLayout.NORTH);

//Designs the layout for the ViewUI.

pack();

setTitle("Finace View");

setSize(**500**, **500**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

/\*\*

\* Adds an action listener to the leaveButton.

\* @param listener the event listener.

\*/

**public** **void** **addLeaveButtonListener**(ActionListener listener) {

leaveButton.addActionListener(listener);

}

**private** Container containerPanel;

**private** JLabel totalLabel;

**private** JPanel historyPanel;

**private** JPanel transactionListPanel;

**private** **final** JButton leaveButton;

}

### LoginViewUI.java

**package** finalProject.views;

**import** **finalProject.models.Store**;

**import** **finalProject.controllers.LoginController**;

**import** **javax.swing.\***;

**import** **java.awt.event.ActionEvent**;

**import** **java.awt.event.ActionListener**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **LoginViewUI** **extends** JFrame {

**private** JLabel usernameLabel;

**private** JLabel passwordLabel;

**private** JTextField usernameTextField;

**private** JPasswordField passwordField;

**private** JButton loginButton;

**private** LoginController loginController;

**public** **LoginViewUI**(Store store) {

// Initialize the UI components

usernameLabel = **new** JLabel("Username:");

passwordLabel = **new** JLabel("Password:");

usernameTextField = **new** JTextField();

passwordField = **new** JPasswordField();

loginButton = **new** JButton("Login");

// Create a new LoginController instance to handle login attempts

loginController = **new** LoginController(**this**, store);

// Set up the UI layout

setLayout(**new** BoxLayout(getContentPane(), BoxLayout.PAGE\_AXIS));

add(usernameLabel);

add(usernameTextField);

add(passwordLabel);

add(passwordField);

add(loginButton);

// Set up the login button action listener

loginButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

String username = usernameTextField.getText();

String password = **new** String(passwordField.getPassword());

// Call the login method of the LoginController to handle the login attempt

loginController.login(username, password);

}

});

// Set up the window properties

setTitle("Login");

setSize(**300**, **150**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

}

### ManagerViewUI.java

**package** finalProject.views;

**import** **finalProject.models.Store**;

**import** **finalProject.controllers.ManagerController**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionEvent**;

**import** **javax.swing.\***;

**import** **java.awt.event.ActionListener**;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **ManagerViewUI** **extends** JFrame {

**private** JButton transactionButton;

**private** JButton stockButton;

**private** JButton financeButton;

**private** ManagerController managerController;

**private** **final** JButton leaveToLoginButton;

**public** **ManagerViewUI**(Store store) {

// Create the buttons and add them to the UI

transactionButton = **new** JButton("View Transaction History");

stockButton = **new** JButton("Stock Shelves");

financeButton = **new** JButton("View Finances");

leaveToLoginButton = **new** JButton("Logout");

// Create a new ManagerController instance to handle button clicks

managerController = **new** ManagerController(**this**, store);

JPanel buttonPanel = **new** JPanel();

buttonPanel.setLayout(**new** GridLayout(**4**, **1**));

buttonPanel.add(transactionButton);

buttonPanel.add(stockButton);

buttonPanel.add(financeButton);

buttonPanel.add(leaveToLoginButton);

// Set up the button action listeners

transactionButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

// Call the browseByAisle method of the DashboardController to handle the button click

managerController.viewTransactionHistory();

}

});

// Set up the button action listeners

stockButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

// Call the browseByAisle method of the DashboardController to handle the button click

managerController.viewStock();

}

});

// Set up the button action listeners

financeButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

// Call the browseByAisle method of the DashboardController to handle the button click

managerController.viewFinances();

}

});

// Set up the button action listeners

leaveToLoginButton.addActionListener(**new** ActionListener() {

**@Override**

**public** **void** **actionPerformed**(ActionEvent e) {

// Call the browseByAisle method of the DashboardController to handle the button click

managerController.logout();

}

});

add(buttonPanel, BorderLayout.CENTER);

//Designs the layout for the ViewUI.

pack();

setTitle("Manager Dashboard");

setSize(**300**, **300**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

}

### SearchViewUI.java

**package** finalProject.views;

**import** **finalProject.models.Store**;

**import** **finalProject.controllers.SearchController**;

**import** **finalProject.models.Item**;

**import** **finalProject.models.ProductIterator**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.Container**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionListener**;

**import** **java.util.ArrayList**;

**import** **javax.swing.\***;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **SearchViewUI** **extends** JFrame {

**private** Store store;

/\*\*

\* Constructor creates a ViewUI for the Search.

\* @param store the Model of the MVC.

\*/

**public** **SearchViewUI**(Store store) {

**this**.store = store;

// Set up the UI layout

setLayout(**new** BoxLayout(getContentPane(), BoxLayout.PAGE\_AXIS));

add(**new** JLabel("Search View"));

//Setup: a Panel object that will contain each of the objects in frame.

searchPanel = **new** JPanel(**new** GridLayout(**4**, **1**));

searchPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

//Setup: Creating the J-Objects within the Panel.

cancelButton = **new** JButton("Cancel");

searchLabel = **new** JLabel("Search for Item:");

searchField = **new** JTextField();

searchButton = **new** JButton("Search");

addToCartButton = **new** JButton("Add to Cart");

itemComboBox = **new** JComboBox();

//Create a new SearchController instance to handle button clicks.

SearchController searchController = **new** SearchController(**this**, store);

//Adds the J-Objects into the Panel.

searchPanel.add(searchLabel);

searchPanel.add(searchField);

searchPanel.add(searchButton);

searchPanel.add(cancelButton);

//Creates the Container.

containerPanel = getContentPane();

containerPanel.add(searchPanel, BorderLayout.NORTH);

// Set up the window properties

setTitle("Search View");

setSize(**300**, **300**);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

/\*\*

\* Adds an action listener to the cancelButton.

\* @param listener the event listener.

\*/

**public** **void** **addCancelButtonListener**(ActionListener listener) {

cancelButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the searchButton.

\* @param listener the event listener.

\*/

**public** **void** **addSearchButtonListener**(ActionListener listener) {

searchButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the searchButton.

\* @param listener the event listener.

\*/

**public** **void** **addItemButtonListener**(ActionListener listener) {

addToCartButton.addActionListener(listener);

}

/\*\*

\* Changes the ViewUI to display a JComboBox containing the items matching

\* inputted string.

\* @param matchingProduct list of matching Items.

\*/

**public** **void** **displayMatchingProducts**(ArrayList<Item> matchingProduct) {

containerPanel.removeAll(); //clears the previous view.

//Setup: a new Panel for selecting the items.

searchPanel = **new** JPanel(**new** GridLayout(**6**, **1**));

searchPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

//Creates an Label for selecting an item.

JLabel itemLabel = **new** JLabel("Select an item:");

//Removes the items from a combo box only if it already has content.

**if** (itemComboBox != **null**){

itemComboBox.removeAllItems();

}

//Adds every product from the Aisle to the JComboBox.

**for** (Item product : matchingProduct) {

itemComboBox.addItem(product);

}

//Creates a Label and TextBox for quantity.

JLabel quantityLabel = **new** JLabel("Enter the quantity:");

quantityField = **new** JTextField();

cancelButton = **new** JButton("Cancel");

//Create a new SearchController instance to handle button clicks.

SearchController searchController = **new** SearchController(**this**, store);

//Adds the J-Objects to the Panel.

searchPanel.add(itemLabel);

searchPanel.add(itemComboBox);

searchPanel.add(quantityLabel);

searchPanel.add(quantityField);

searchPanel.add(addToCartButton);

searchPanel.add(cancelButton);

//Resets the container with the new Panel.

containerPanel.add(searchPanel, BorderLayout.NORTH);

containerPanel.revalidate();

containerPanel.repaint(); //Repaints the container.

}

**private** Container containerPanel;

**private** JPanel searchPanel;

**private** **final** JLabel searchLabel;

**private** JButton cancelButton;

**private** **final** JButton searchButton;

**private** **final** JButton addToCartButton;

**public** JTextField searchField;

**public** JTextField quantityField;

**public** JComboBox<Item> itemComboBox;

}

### StockViewUI.java

**package** finalProject.views;

**import** **finalProject.models.Aisle**;

**import** **finalProject.controllers.StockController**;

**import** **finalProject.models.Item**;

**import** **finalProject.models.Store**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.Container**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionListener**;

**import** **javax.swing.\***;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **StockViewUI** **extends** JFrame {

/\*\*

\* Constructor creates a ViewUI for the Aisle.

\* @param store the Model of the MVC.

\*/

**public** **StockViewUI**(Store store) {

//Setup: Creates a Window for the AisleView.

**super**("Aisle View");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

**this**.store = store; //Conencts the store to the AisleViewUI.

//Setup: a Panel object that will contain each of the objects in frame.

aislePanel = **new** JPanel(**new** GridLayout(**3**, **1**));

aislePanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

//Setup: Creating the J-Objects within the Panel.

cancelButton = **new** JButton("Cancel");

aisleLabel = **new** JLabel("Select an aisle number:");

aisleNumberField = **new** JTextField();

aisleButton = **new** JButton("Select Aisle");

addToCartButton = **new** JButton("Add to Cart");

itemComboBox = **new** JComboBox();

//Create a new StockController instance to handle button clicks.

StockController stockController = **new** StockController(**this**, store);

//Adds the J-Objects into the Panel.

aislePanel.add(aisleLabel);

aislePanel.add(aisleNumberField);

aislePanel.add(aisleButton);

aislePanel.add(cancelButton);

//Creates the Container.

containerPanel = getContentPane();

containerPanel.add(aislePanel, BorderLayout.NORTH);

//Designs the layout for the ViewUI.

pack();

setTitle("Aisle View");

setSize(**300**, **300**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

/\*\*

\* Adds an action listener to the aisleButton.

\* @param listener the event listener.

\*/

**public** **void** **addAisleButtonListener**(ActionListener listener) {

aisleButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the addToCartButton.

\* @param listener the event listener.

\*/

**public** **void** **addItemButtonListener**(ActionListener listener) {

addToCartButton.addActionListener(listener);

}

/\*\*

\* Adds an action listener to the cancelButton.

\* @param listener the event listener.

\*/

**public** **void** **addCancelButtonListener**(ActionListener listener) {

cancelButton.addActionListener(listener);

}

/\*\*

\* Displays a different Aisle view that contains an item picker.

\* @param aisleNumber the aisle that is being looked into.

\*/

**public** **void** **selectAisle**(**int** aisleNumber) {

containerPanel.removeAll(); //clears the previous view.

//Setup: a new Panel for selecting the items.

itemPanel = **new** JPanel(**new** GridLayout(**6**, **1**));

itemPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

//Creates an Label for selecting an item.

JLabel itemLabel = **new** JLabel("Select an item:");

Aisle currentAisle = store.getAisle(aisleNumber);

JLabel aisleLabel = **new** JLabel(currentAisle.getAisleName());

//Removes the items from a combo box only if it already has content.

**if** (itemComboBox != **null**){

itemComboBox.removeAllItems();

}

//Adds every product from the Aisle to the JComboBox.

**for** (Item product : currentAisle.getItems()) {

itemComboBox.addItem(product);

}

//Creates a Label and TextBox for quantity.

JLabel quantityLabel = **new** JLabel("Enter the quantity:");

quantityField = **new** JTextField();

//Adds the J-Objects to the Panel.

itemPanel.add(aisleLabel);

itemPanel.add(itemLabel);

itemPanel.add(itemComboBox);

itemPanel.add(quantityLabel);

itemPanel.add(quantityField);

itemPanel.add(addToCartButton);

//Resets the container with the new Panel.

containerPanel.add(itemPanel, BorderLayout.NORTH);

containerPanel.revalidate();

containerPanel.repaint(); //Repaints the container.

}

/\*\*

\* Adds the selected Item to the current users Cart.

\*/

**public** **void** **addToCart**() {

Item selectedItem = (Item) itemComboBox.getSelectedItem();

**int** quantity = Integer.parseInt(quantityField.getText());

**for** (**int** i=**0**; i<quantity; i++) {

store.currentAccount.addToCart(selectedItem);

}

}

/\*\*

\* Get the aisle number entered by the user.

\* @return the number entered into the text field.

\*/

**public** **int** **getAisleNumber**() {

**return** Integer.parseInt(aisleNumberField.getText());

}

**private** **final** Store store;

**private** Container containerPanel;

**private** JPanel aislePanel;

**private** JPanel itemPanel;

**private** **final** JLabel aisleLabel;

**public** JComboBox itemComboBox;

**private** JTextField aisleNumberField;

**private** **final** JButton aisleButton;

**private** **final** JButton cancelButton;

**public** JTextField quantityField;

**private** **final** JButton addToCartButton;

}

### TransactionViewUI.java

**package** finalProject.views;

**import** **finalProject.controllers.TransactionController**;

**import** **finalProject.models.Transaction**;

**import** **finalProject.models.Store**;

**import** **java.awt.BorderLayout**;

**import** **java.awt.Container**;

**import** **java.awt.GridLayout**;

**import** **java.awt.event.ActionListener**;

**import** **javax.swing.\***;

/\*\*

\*

\* @author Alex Reburn

\*/

**public** **class** **TransactionViewUI** **extends** JFrame {

**public** **TransactionViewUI**(Store store) {

//Setup: Creates a Window for the CheckoutViewUI.

historyPanel = **new** JPanel(**new** GridLayout(**5**, **1**));

historyPanel.setBorder(BorderFactory.createEmptyBorder(

**10**, **10**, **10**, **10**));

transactionListPanel = **new** JPanel(**new** GridLayout(**5**, **1**));

//creates a JLabel for all of the Items.

**for** (Transaction t : store.getTransactions()) {

transactionListPanel.add(**new** JLabel("Account: " + t.getName()

+ ", Money: " + t.getTotal()));

}

//Setup: Creating the J-Objects within the Panel.

leaveButton = **new** JButton("Leave");

//Create a new CheckoutController instance to handle button clicks

TransactionController transactionController = **new** TransactionController(**this**, store);

//Adds the J-Objects into the Panel.

historyPanel.add(transactionListPanel);

historyPanel.add(leaveButton);

//Creates the Container.

containerPanel = getContentPane();

containerPanel.add(historyPanel, BorderLayout.NORTH);

//Designs the layout for the ViewUI.

pack();

setTitle("Transaction View");

setSize(**500**, **500**);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

/\*\*

\* Adds an action listener to the leaveButton.

\* @param listener the event listener.

\*/

**public** **void** **addLeaveButtonListener**(ActionListener listener) {

leaveButton.addActionListener(listener);

}

**private** Container containerPanel;

**private** JPanel historyPanel;

**private** JPanel transactionListPanel;

**private** **final** JButton leaveButton;

}