

# Shopping Cart

Jared Usher, James Berry, Kirsten Hernquist

## Revised Functional Specifications with GUI

<u>Requirement ID</u>	<u>Requirement Statement</u>	<u>Must/Want</u>	<u>Comments</u>
FR.001	The application shall have a homepage that displays the purpose.	Must	Kirsten Hernquist
FR.002	Upon arrival at the homepage, the app shall provide a window for login.	Must	Kirsten Hernquist
FR.003	The app shall provide a shopping cart for purchases.	Must	Kirsten Hernquist
FR.004	The app shall provide a list of available products.	Must	Kirsten Hernquist
FR.005	The app shall verify that the product description is valid.	Must	Kirsten Hernquist
FR.006	The system shall store cart contents.	Must	Kirsten Hernquist
FR.007	The system shall verify login information.	Must	Kirsten Hernquist
FR.008	The system shall store login information.	Must	Kirsten Hernquist
FR.009	The app shall provide a 'favorite' button on every product listing.	Must	Kirsten Hernquist
FR.010	Users are available to view their list of favorite products.	Want	Jared Usher
FR.011	Users are able to unfavorite products on their list of favorite products.	Want	Jared Usher
FR.012	On a product listing, Users can add items to the shopping card, depending on the quantity currently available.	Must	Jared Usher
FR.013	The user can proceed to the checkout window at any time by clicking the 'checkout' button.	Must	Jared Usher
FR.014	On the checkout window, the shopping cart can be updated by changing the	Must	Jared Usher

	item count for each product in the cart by clicking the '+' or '-' button.		
FR.015	The user enters their payment information and confirms their order by clicking the 'confirm' button to complete checkout.	Must	Jared Usher
FR.016	The app stores the previous orders of a user (serialize).	Want	Jared Usher
FR.017	Users are able to view all of their own previous orders (serialize).	Want	Jared Usher
FR.018	Sellers are able to access their seller inventory after login.	Must	Jared Usher
FR.019	Seller inventory displays listed products, their information, and their current available quantity.	Must	Jared Usher
FR.020	Sellers can update the quantity available of each product in their inventory by clicking the '+' button.	Must	Jared Usher
FR.021	Sellers are able to add new products to their inventory by clicking the 'add product' button.	Must	Jared Usher
FR.022	Quantity of product available automatically drops when product is bought	Must	James
FR.023	User can browse and search available products	Must	James

**OS:** Mac/Windows/Linux

## Revised Use Cases with GUI

### 1. Log in (James)

- a. User selects 'log in'
- b. System displays login form
- c. User enters username and password
- d. System verifies login

#### Use case 1B: Failed login

- a. Scenario starts at UC1 step (c), when user enters mismatching username and password, or username is not in system
- b. System displays failed login message
  - i. "Username and password do not match"
- c. System waits a few seconds(prevent brute force password guessing?)
- d. System returns to UC1 step (b)

### 2. Seller lists an item for sale (James)

- a. Seller carries out **Log in**
- b. Seller selects 'add new listing'
- c. Seller inputs information for new item
- d. System updates seller inventory and listings with new item

### 3. Buyer adds item to cart (James)

- a. Buyer carries out **Log in**
- b. Buyer selects an item
- c. Buyer selects 'add to cart'
- d. System saves item to user's cart list

### 4. Buyer checks out cart (Jared)

- a. Buyer clicks 'checkout'
- b. System retrieves list of products in Buyer's cart
- c. System displays multiple options:
  - i. Modify Product Amount In Cart
  - ii. Checkout
  - iii. Cancel
- d. Buyer selects 'checkout'
- e. System displays list of products again
- f. Buyer selects "confirm" order
- g. Buyer enters payment information
- h. System adds order to Buyer's order list

#### Variation of 4: Canceled Checkout

- a. Scenario starts from UC 4 step c

- b. Buyer selects 'Cancel'
- c. System displays browseable list of products

- 5. Buyer browses sale items(James)
  - a. Buyer selects "search"
  - b. System displays search form
  - c. Buyer inputs search data
  - d. System displays item list matching search input

#### 6. Seller updates inventory (Kirsten)

- a. Seller carries out **Log in**
- b. System retrieves (active) current inventory list and displays it on the seller homepage window
- c. Seller modifies (adds/removes) item counts via the '+/-' buttons.
- d. System saves updated list.

#### Variation of 6: (Kirsten)

- a. Scenario starts from UC 6 step k.
- b. Seller modifies item counts incorrectly.
- c. System saves updated list.
- d. Seller returns to step 6k.
- e. Seller modifies item counts.
- f. System saves updated list.

- 7. Buyer adds item to favorites list (Kirsten)
  - a. Buyer carries out **Log in**.
  - b. Browse window appears.
  - c. Buyer clicks an item via the 'Product Name' button.
  - d. App displays product details in pop-up window.
  - e. User clicks 'add to favorites' button.
  - f. System updates favorites list.

#### Variation 7a: (Kirsten)

- a. Scenario starts from UC 7 step e.
- b. Buyer hit's 'add to cart' button by accident.
- c. System updates the cart list.
- d. Buyer clicks 'checkout' button.
- e. Checkout window pops up.
- f. Buyer deletes item from cart by clicking the '-' button until the quantity reaches 0.
- g. System updates cart.
- h. Buyer clicks 'continue shopping' button.
- i. Return to UC 7b.

Variation 7b: (Kirsten)

- a. Scenario starts from UC 7 step e.
- b. User clicks 'add to cart' by accident.
- c. System updates cart list.
- d. User returns to browse window and continues shopping.
- e. User clicks 'check out'.
- f. System displays checkout window.
- g. User deletes item from cart.
- h. System updates cart list.
- i. Return to UC 7b.

8. Buyer views product description (Jared)

- a. Buyer carries out **Log in**
- b. Buyer clicks an item
- c. System retrieves product details (i.e. full product description, pricing and availability)
- d. System displays product details (in popup window)

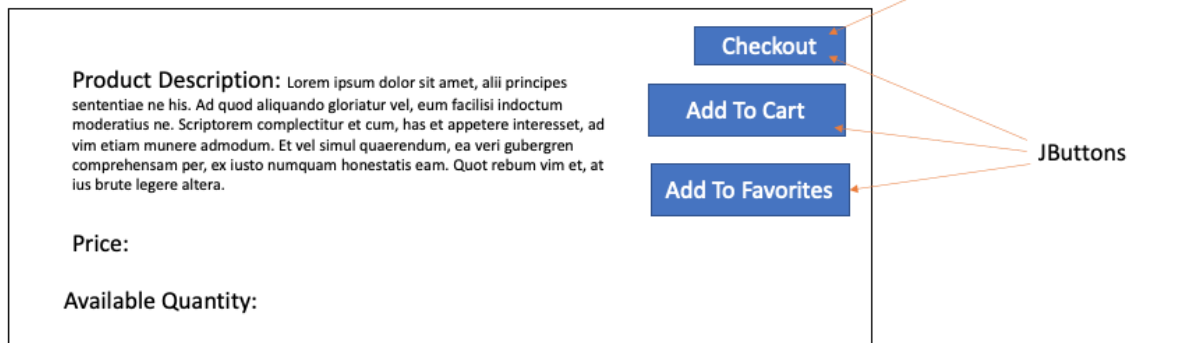
9. Buyer views order status/details (Jared)

- a. Buyer carries out **Log in**
- b. Buyer clicks 'view orders' button
- c. System retrieves list of Buyer's past orders
- d. App displays order details (i.e, products ordered, quantity ordered, price)

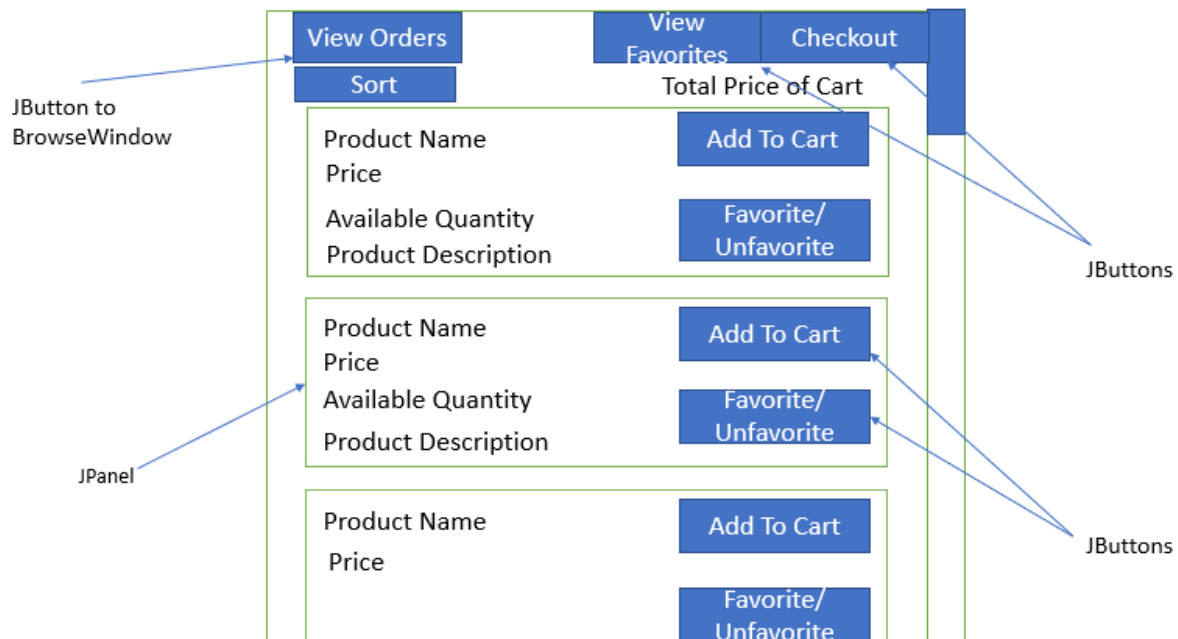
Pictures of GUI  
(favorites window, item listing popup)

## ProductPopUpWindow

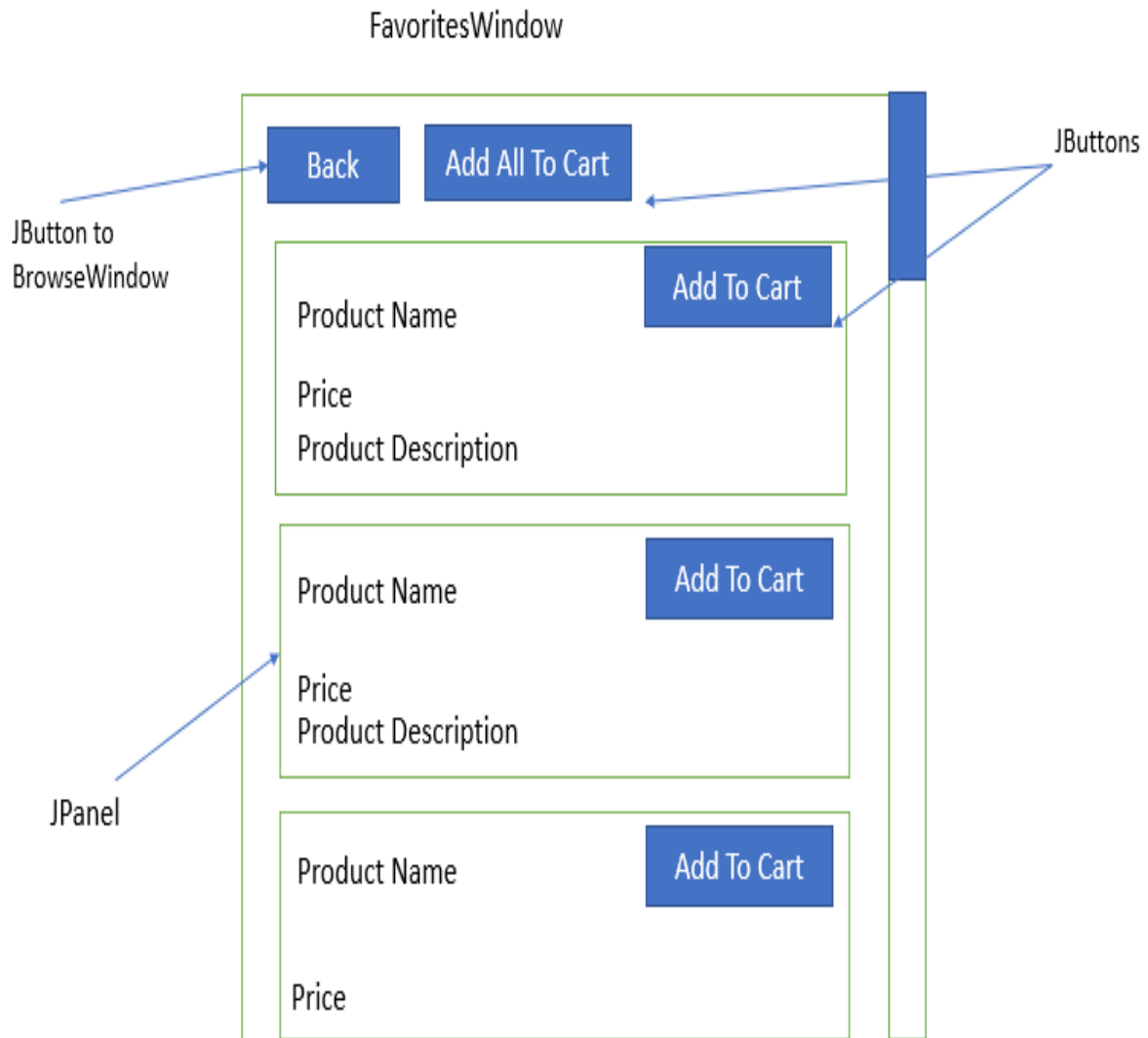
Product **Pop-Up Window** (ProductWindow)



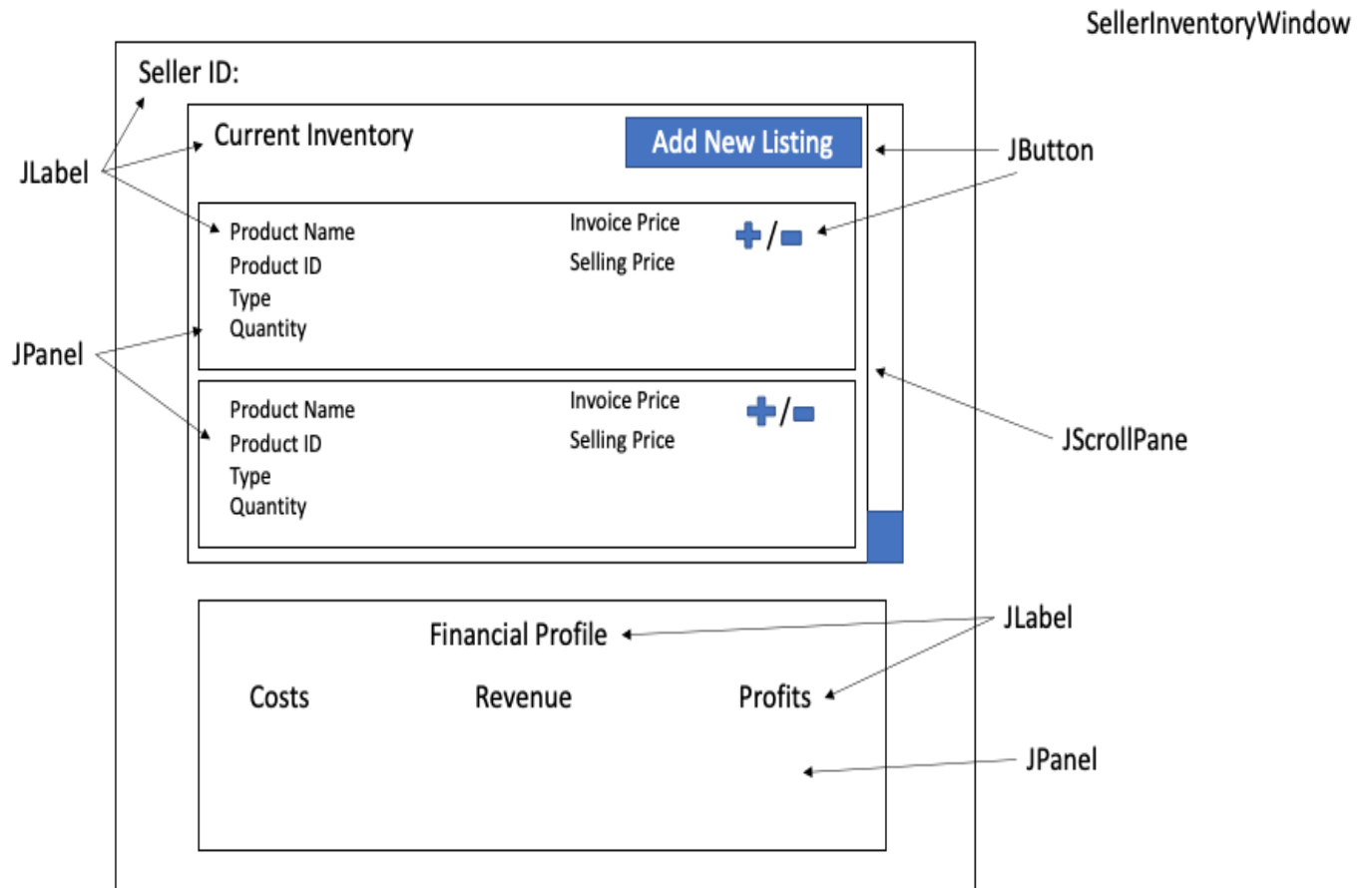
## BrowseWindow



## **FavoritesWindow**



## SellerInventoryWindow





## CheckoutWindow

JScrollPane

JButton

Check out cart

Product name

Price: \$x

Quantity:◀ x ▶

Product name

Price: \$x

Quantity:◀ x ▶

Product name

Price: \$x

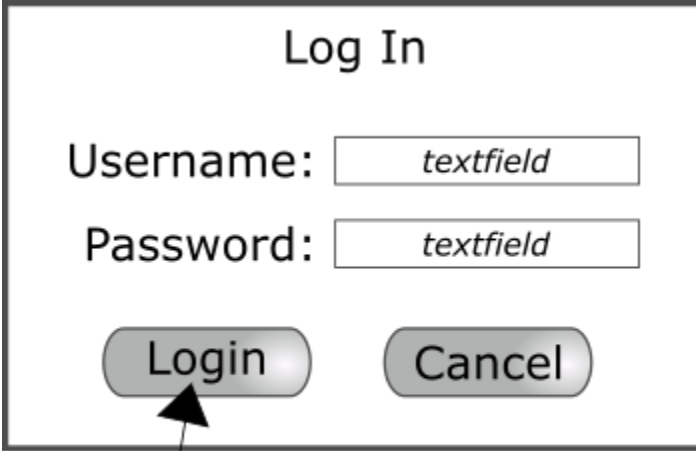
Quantity:◀ x ▶

Total: \$x

Order

Cancel Checkout

### LoginWindow



Log In

Username:

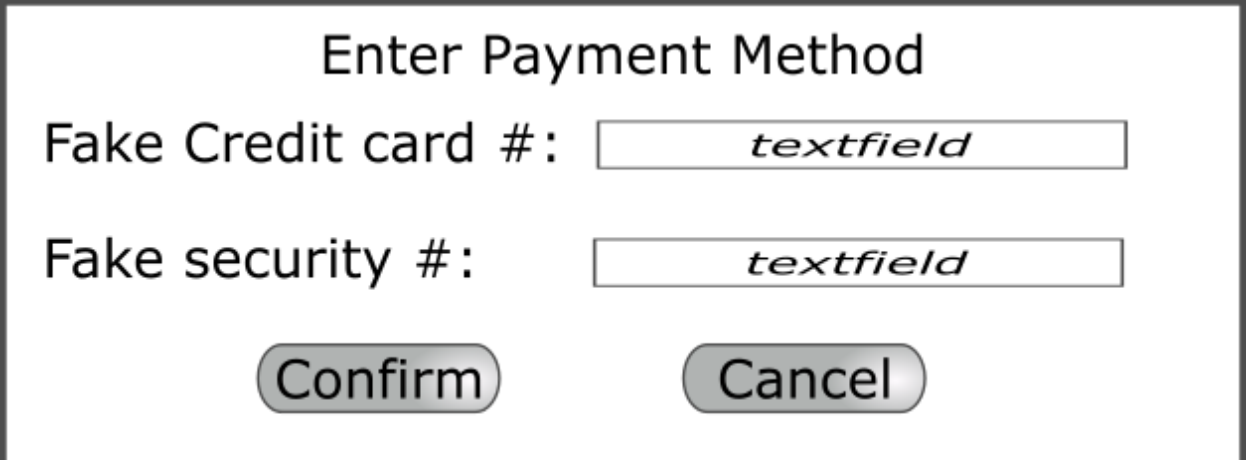
Password:

Login Cancel

JButton

The diagram shows a rectangular window titled "Log In". Inside, there are two labels, "Username:" and "Password:", each followed by a rectangular text field containing the word "textfield". Below these are two rounded rectangular buttons labeled "Login" and "Cancel". An arrow points from the text "JButton" below the window to the "Login" button.

### PaymentWindow



Enter Payment Method

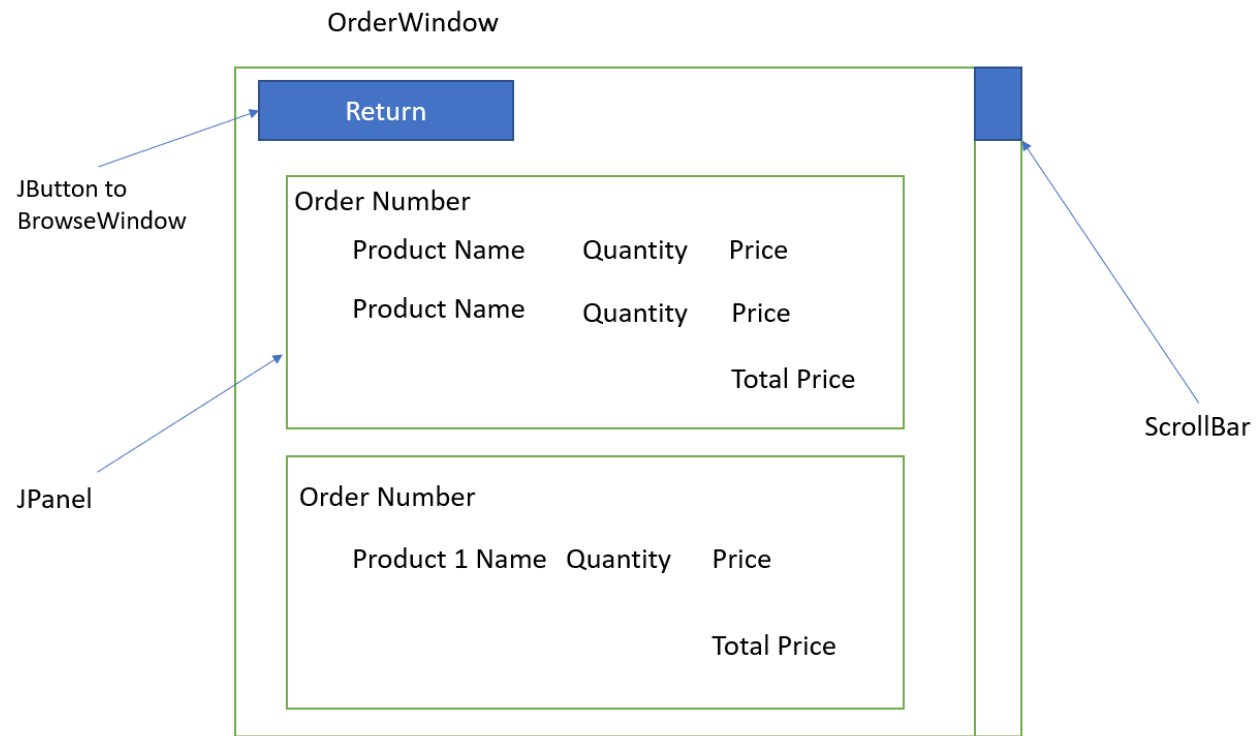
Fake Credit card #:

Fake security #:

Confirm Cancel

The diagram shows a rectangular window titled "Enter Payment Method". Inside, there are two labels, "Fake Credit card #:" and "Fake security #:", each followed by a rectangular text field containing the word "textfield". Below these are two rounded rectangular buttons labeled "Confirm" and "Cancel".

OrderWindow



## **Classes**

UserDB  
Buyer  
Seller  
ShoppingCart  
Product  
DiscountedProduct  
SingletonProductList  
Order  
OrderList  
FavoritesList  
FavoritesBundle  
SellerInventory  
SellerFinancialView  
SortByName  
SortStrategy  
SortByPrice  
  
BrowseWindow  
CheckoutWindow  
FavoritesWindow  
LoginWindow  
NewItemWindow  
OrderWindow  
PaymentWindow  
ProductPopupWindow  
SellerInventoryWindow

## **CRC Cards**

### **UserDB**

Responsibilities:

- Manage user list, passwords
- Access login information
- Stores accounts

Collaborators:

- Buyer
- Seller

- BrowseWindow
- SellerInventoryWindow

### **Buyer**

Responsibilities:

- Manage User Information

Collaborators:

### **Seller**

Responsibilities:

- Manage User Information

Collaborators:

### **ShoppingCart (Model)**

Responsibilities:

- Manage list of products in cart
- Calculate total cost of contents

Collaborators:

- Product
- Buyer

### **Product (Model)**

Responsibilities:

- Manage Contents (id, desc, sell price, quantity)

Collaborators:

### **DiscountedProduct**

Responsibilities:

- Apply a discount to select products

Collaborators:

### **SingletonProductList (Model)**

Responsibilities:

- Holds browseable list of product listings
- Save current state of list
- Notifies observers of changes to inventory
- Get product information and pass along
- Manage contents

Collaborators:

- Product

## **Order**

Responsibilities:

- Manage Contents (products purchased, order price)

Collaborators:

## **OrderList**

Responsibilities:

- Manage contents (add order)

Collaborators:

- Buyer

## **FavoritesList (Model)**

Responsibilities:

- Manage contents (add or remove product)

Collaborators:

- SingletonProductList
- Buyer

## **FavoritesBundle**

Responsibilities:

- Manage contents (add favorite products)

Collaborators:

- FavoritesList

## **SellerInventory**

Responsibilities:

- Contain items for sale by seller
- Notify list of product of changes in inventory
- Manage contents
- Calculate and save costs, revenues, and profits.
- Manage sold item list
- Save current state of inventory

Collaborators:

- SingletonProductList
- SellerInventoryWindow
- Seller

## **BrowseWindow**

Responsibilities:

- Display list of active sale items and searched items
- Display cost of item in cart
- Provide access to checkout, favorites, and orders

Collaborators:

- SingletonProductList
- ProductPopUpWindow
- ShoppingCart
- CheckoutWindow
- FavoritesWindow

### **CheckoutWindow (Controller)**

Responsibilities:

- Display shopping cart contents
- Display option to modify shopping cart contents
- Display window for payment information on click of 'checkout' button
- Update seller Inventory on checkout

Collaborators:

- ShoppingCart
- PaymentWindow
- SingletonProductList
- SellerInventory

### **FavoritesWindow:**

Responsibilities:

- Display favorites list of a buyer

Collaborators:

- FavoritesList

### **LoginWindow**

Responsibilities:

- Display form for user login
- Submit given information to system
- Notify user if username/password is incorrect

Collaborators:

- UserDB

### **NewItemWindow**

Responsibilities:

- Display form for entering new product

Collaborators:

- Seller
- Product

### **OrderWindow**

Responsibilities:

- Display past orders of a buyer

Collaborators:

- OrderList

### **PaymentWindow**

Responsibilities:

- Display form for entering payment information
- Submit payment information

Collaborators:

### **ProductPopUpWindow**

Responsibilities:

- Display all information about a product

Collaborators:

- Product

### **SellerFinancialView (view)**

Responsibilities:

- Display seller costs, revenue, profits
- Update numbers according to seller activity

Collaborators:

### **SellerInventoryWindow**

Responsibilities:

- Display current state of seller's inventory
- Allow for adding products to inventory
- Allow for modifying available quantities of products
- Allow for editing of product information
- Display costs, revenues, and profits

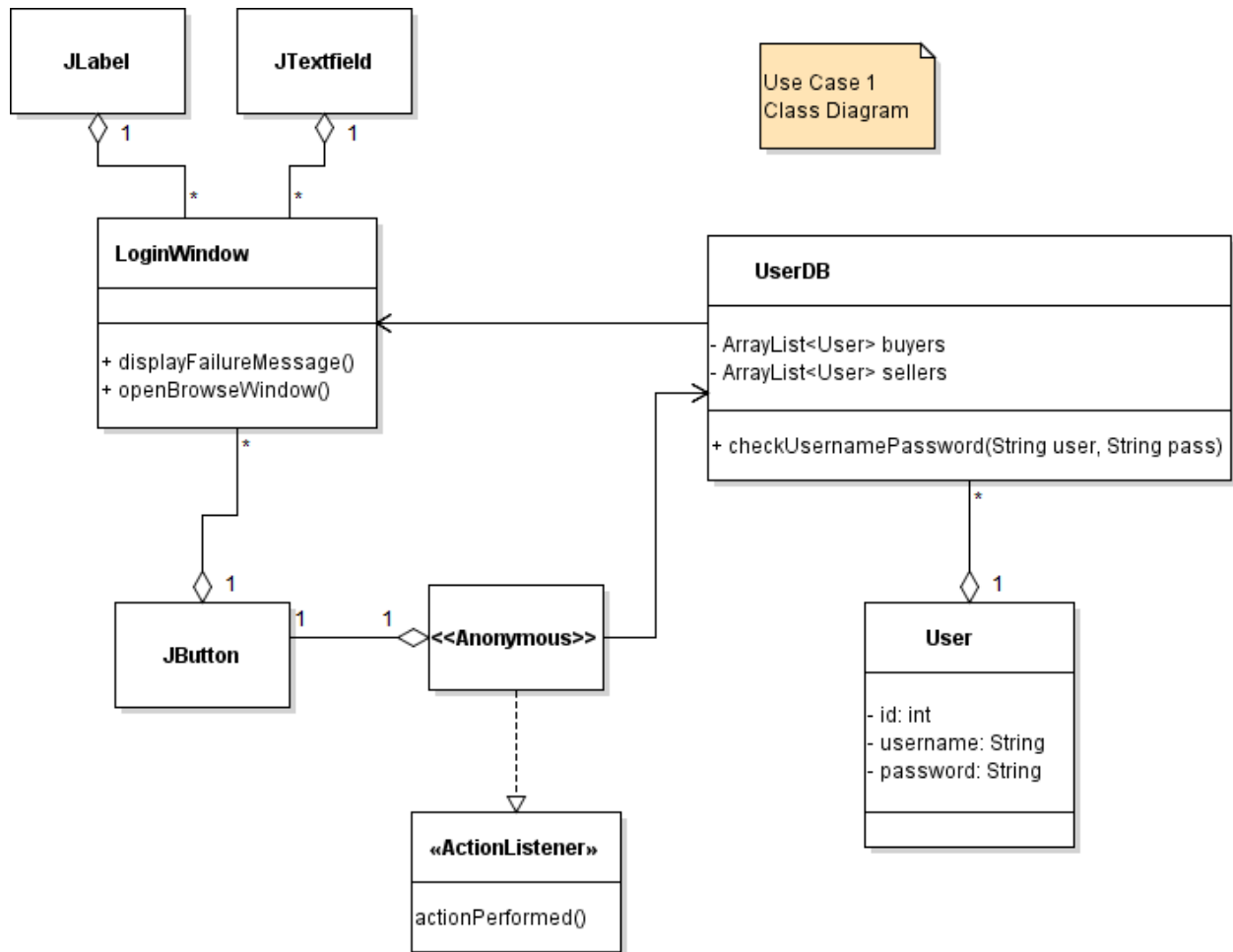
Collaborators:

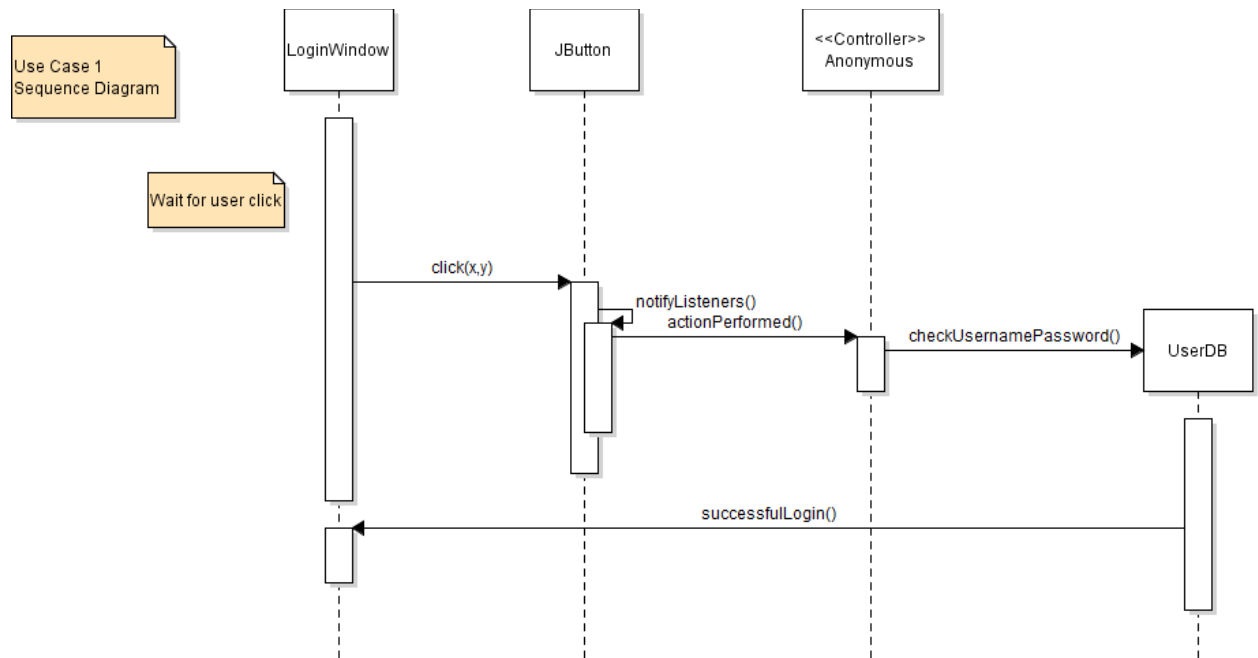
- SellerInventory
- BrowseWindow
- NewItemWindow



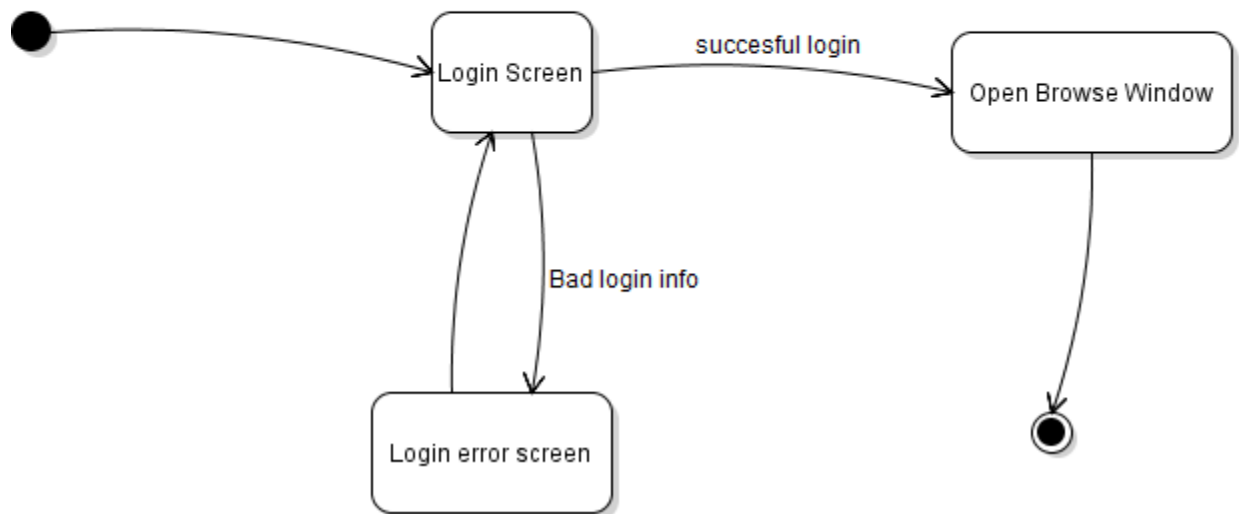
# UML Diagrams

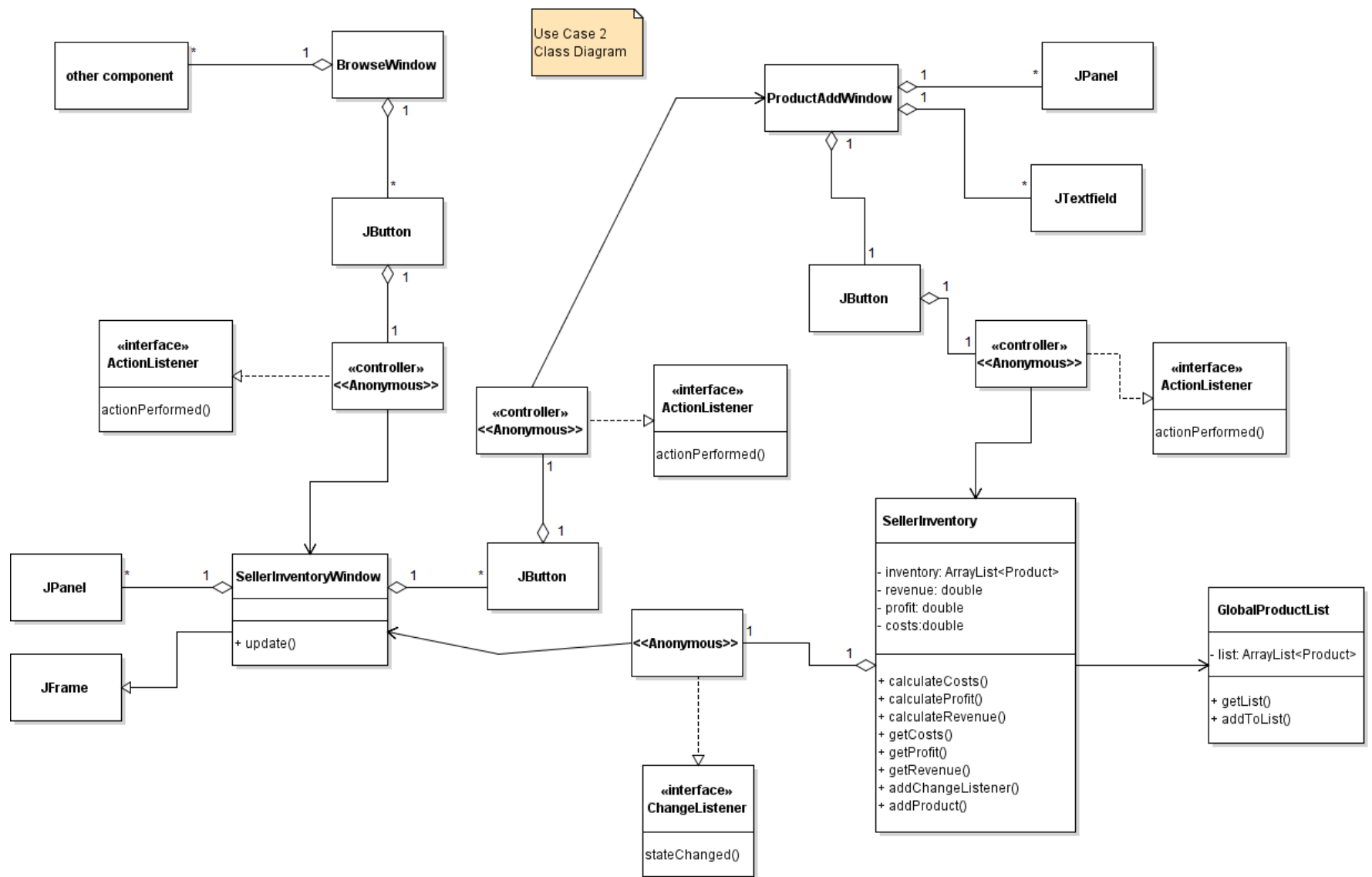
UC 1 & 2

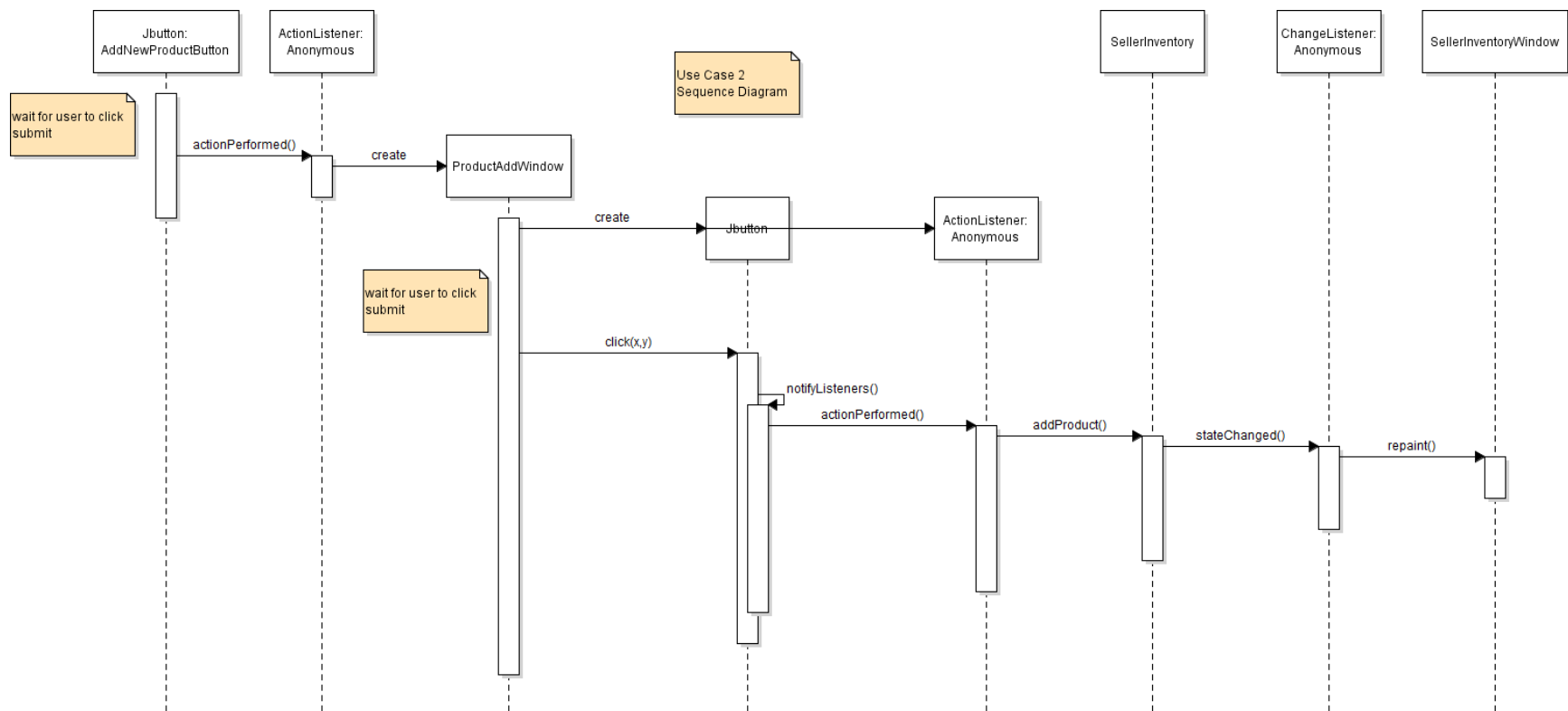




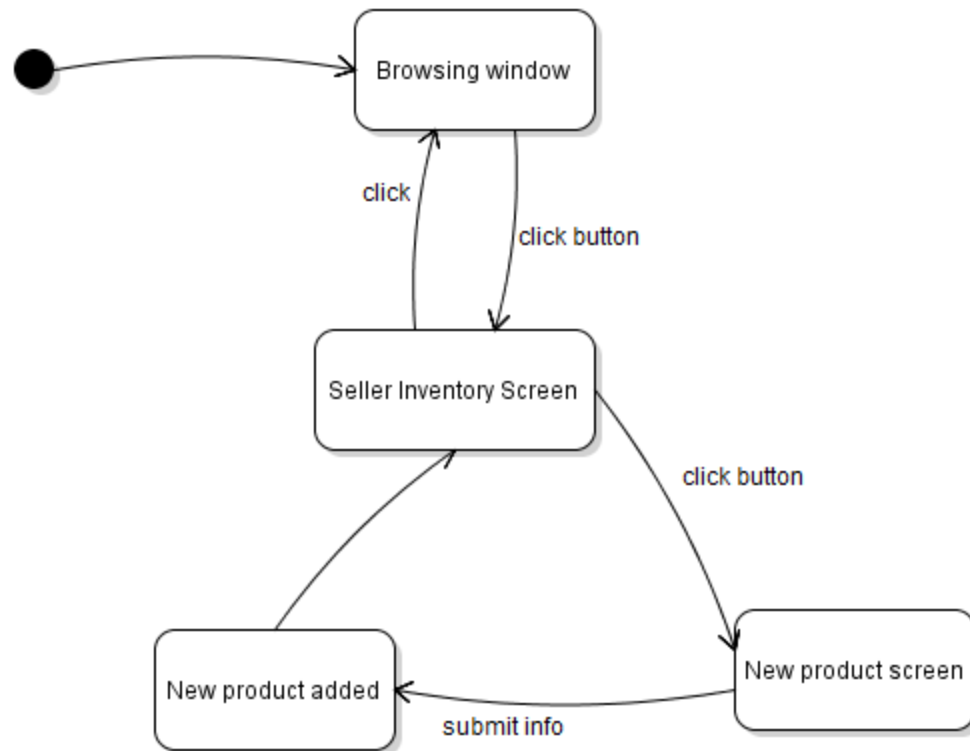
Use case 1  
State Diagram



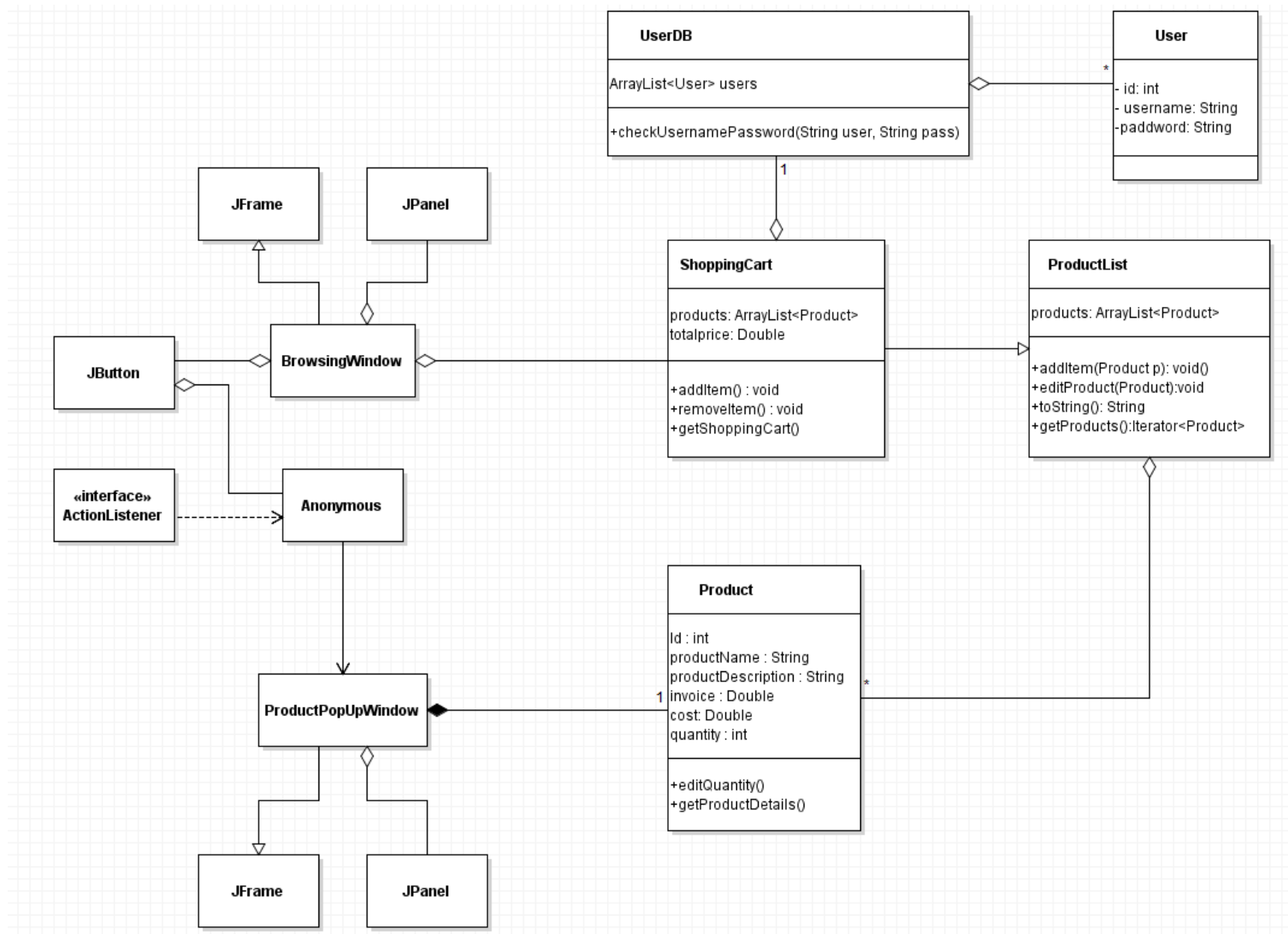


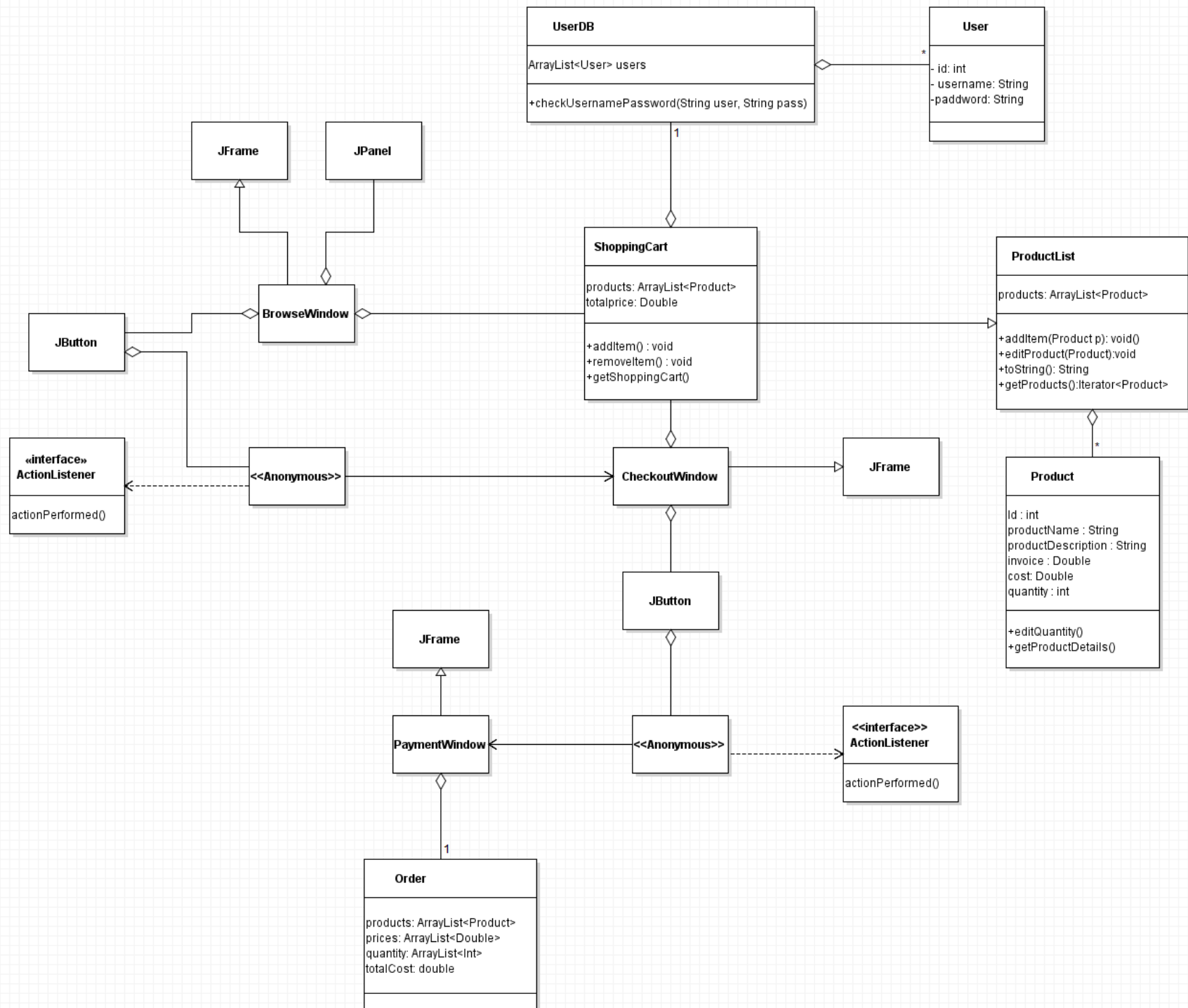


Use case 2  
State Diagram



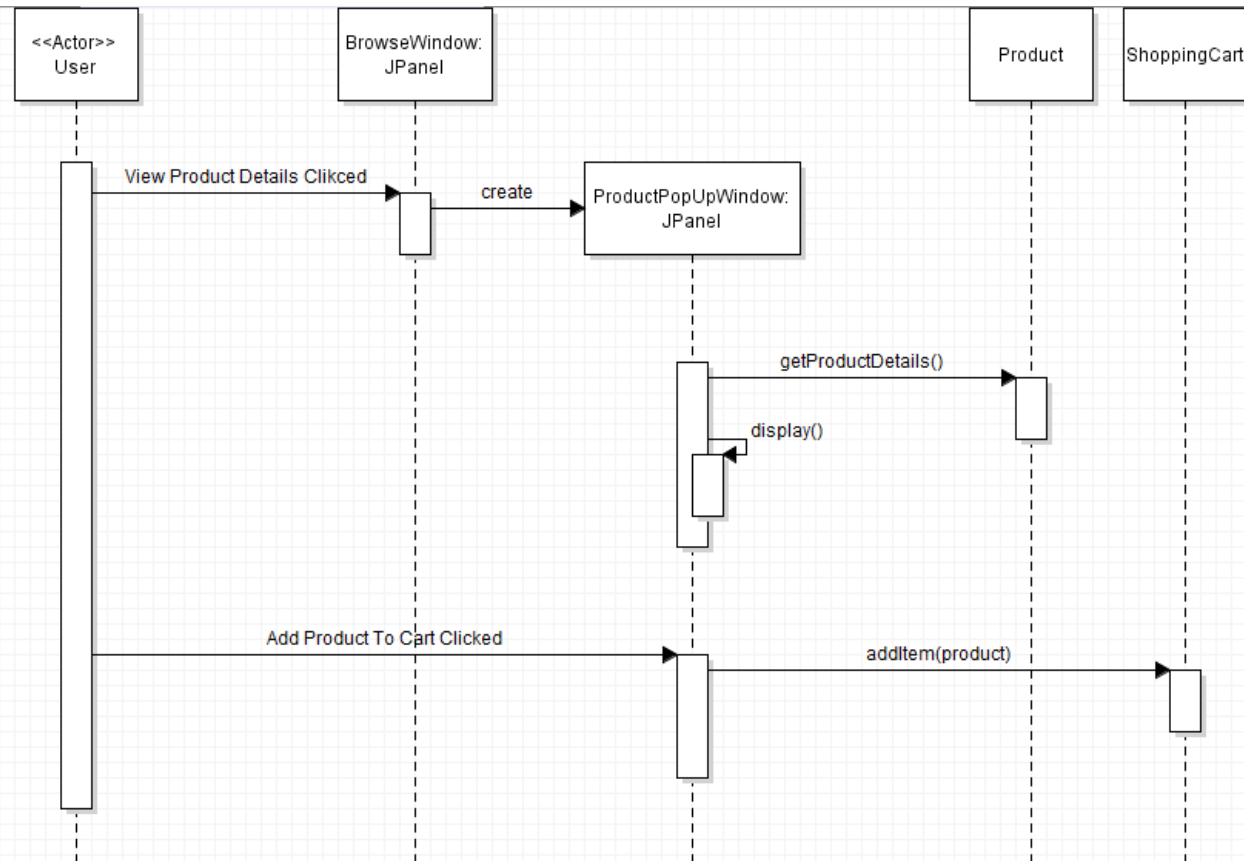
UC 3 & 4





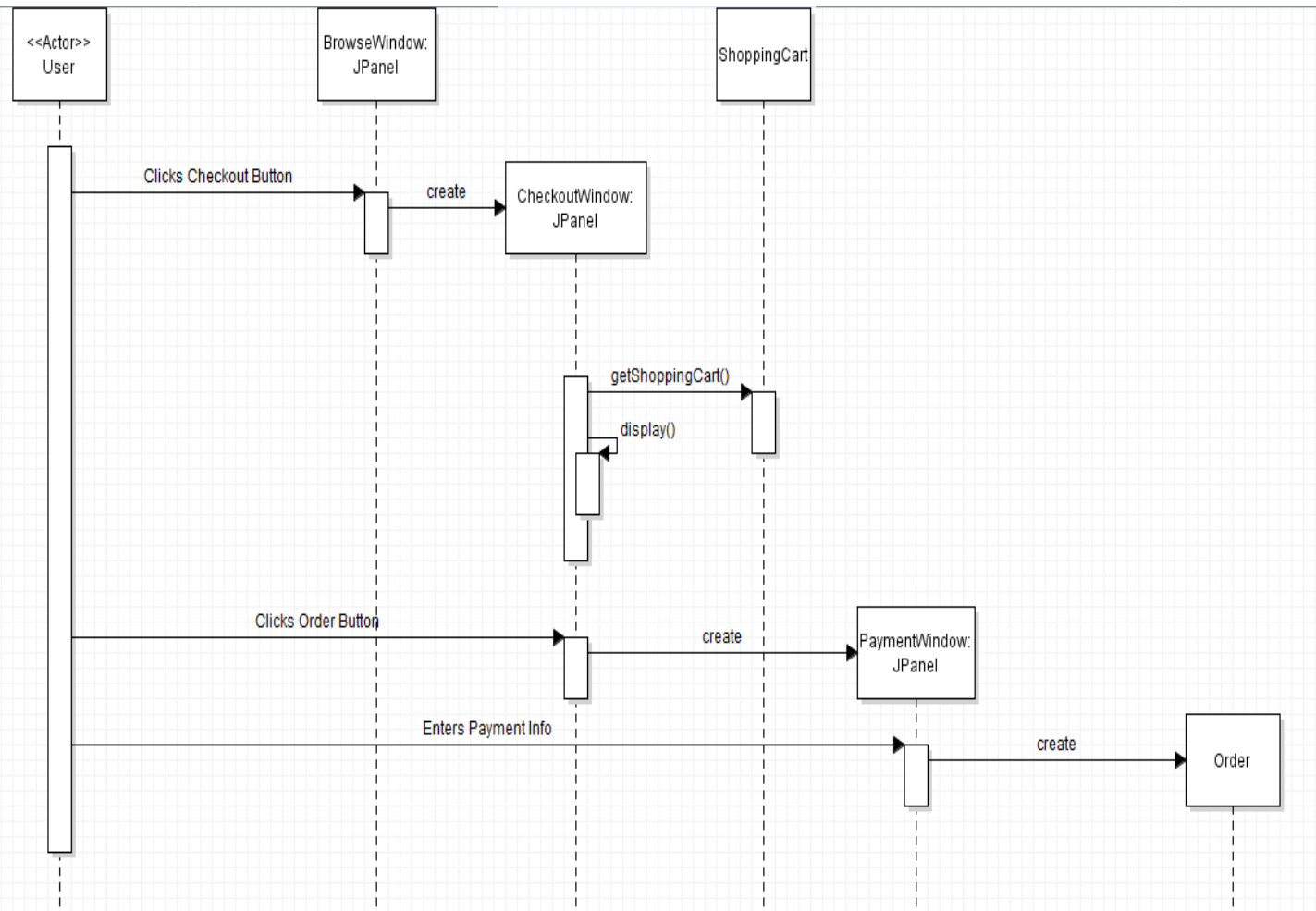


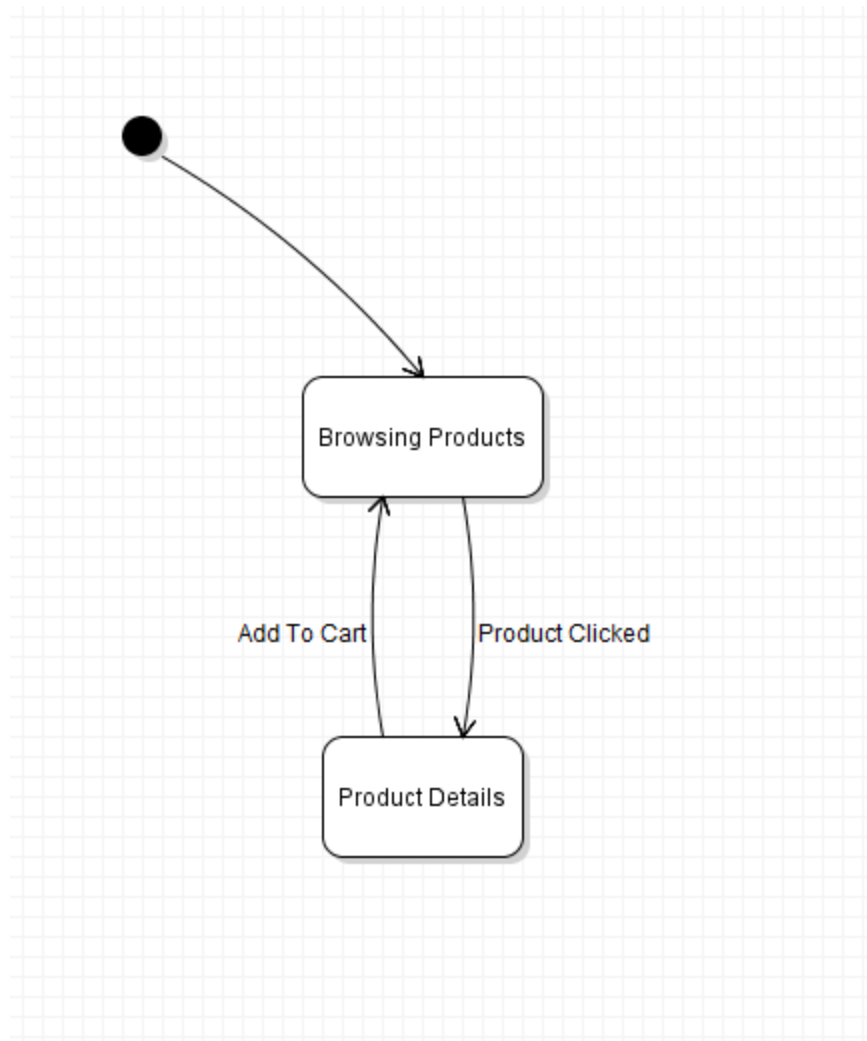
Use Case 3: Buyer Adds Item To Cart  
a) Buyer carries out Log in  
b) Buyer selects an item  
c) Buyer selects 'add to cart'  
d) System saves item to user's cart list

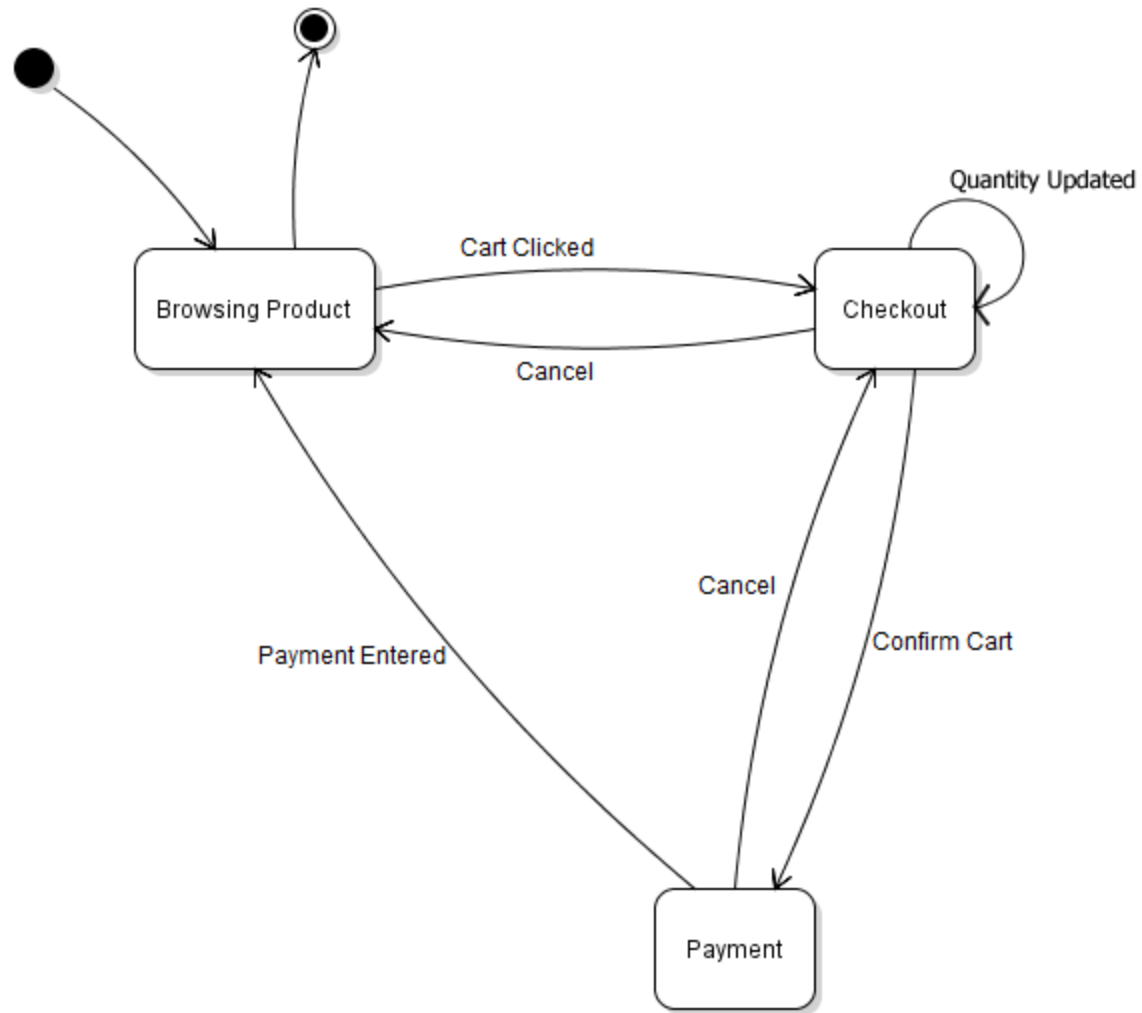


Use Case 4: Buyer Checks Out Cart

- a) Buyer clicks 'checkout'
- b) System retrieves list of products in Buyer's cart
- c) System displays multiple options:
  - Modify Product Amount In Cart
  - Checkout
  - Cancel
- d) Buyer selects 'checkout'
- e) System displays list of products again
- f) Buyer selects "confirm" order
- g) Buyer enters payment information
- h) System adds order to Buyer's order list



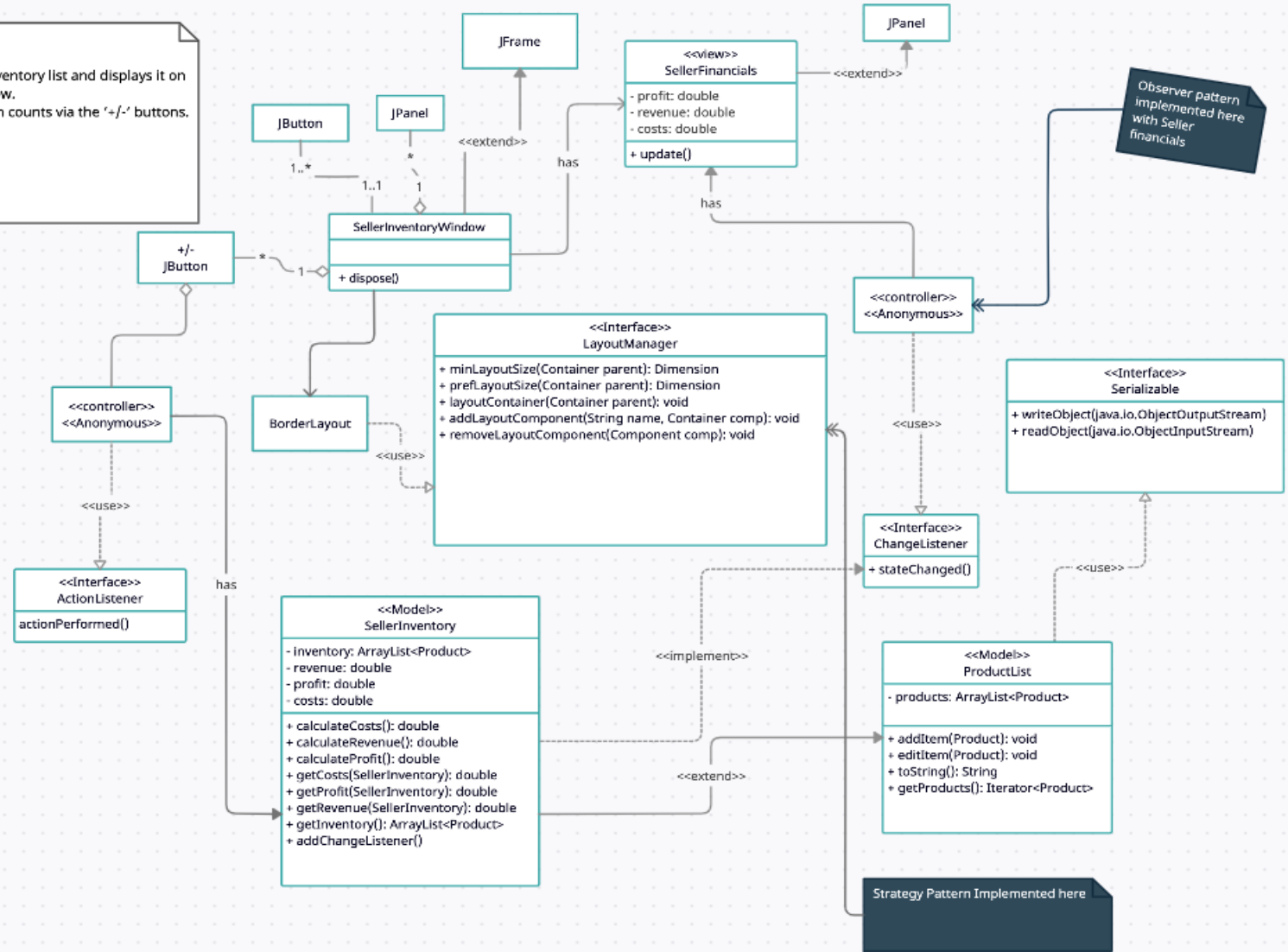




UC 6 & 9

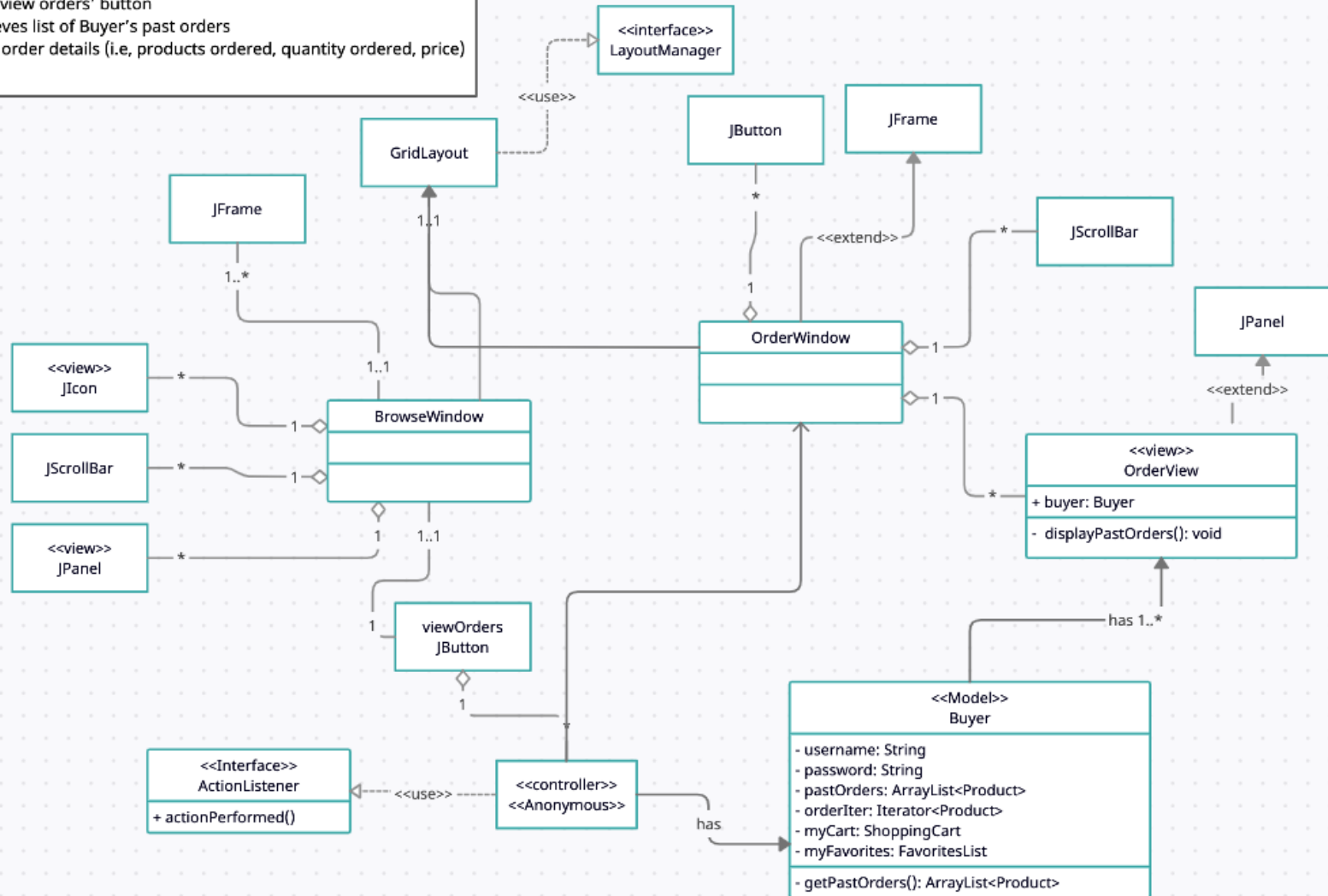
Seller updates inventory  
 Seller carries out Log in  
 System retrieves (active) current inventory list and displays it on entry to the seller homepage window.  
 Seller modifies (adds/removes) item counts via the '+/-' buttons.  
 System saves updated list.

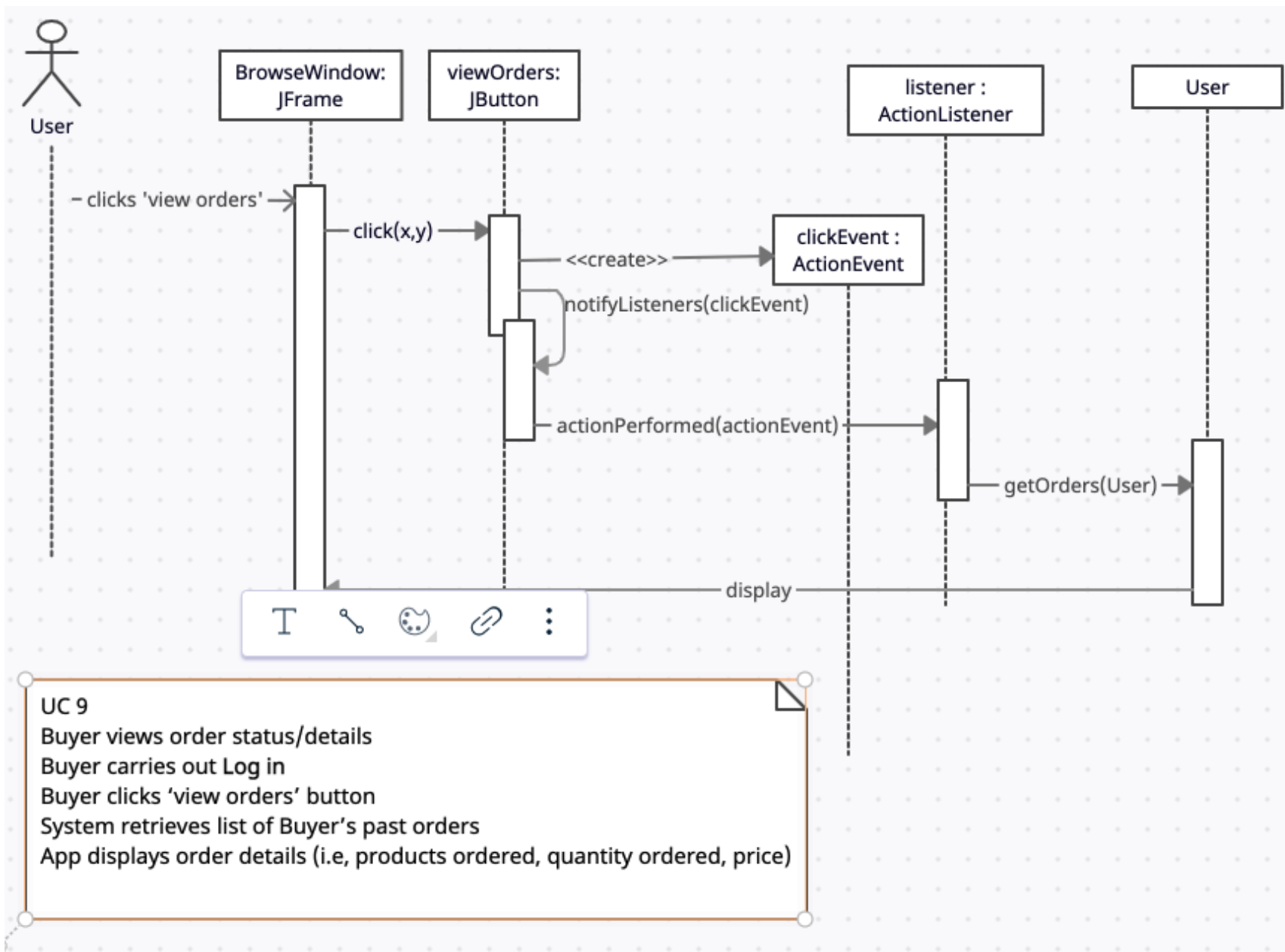
Observer pattern implemented here with Seller financials



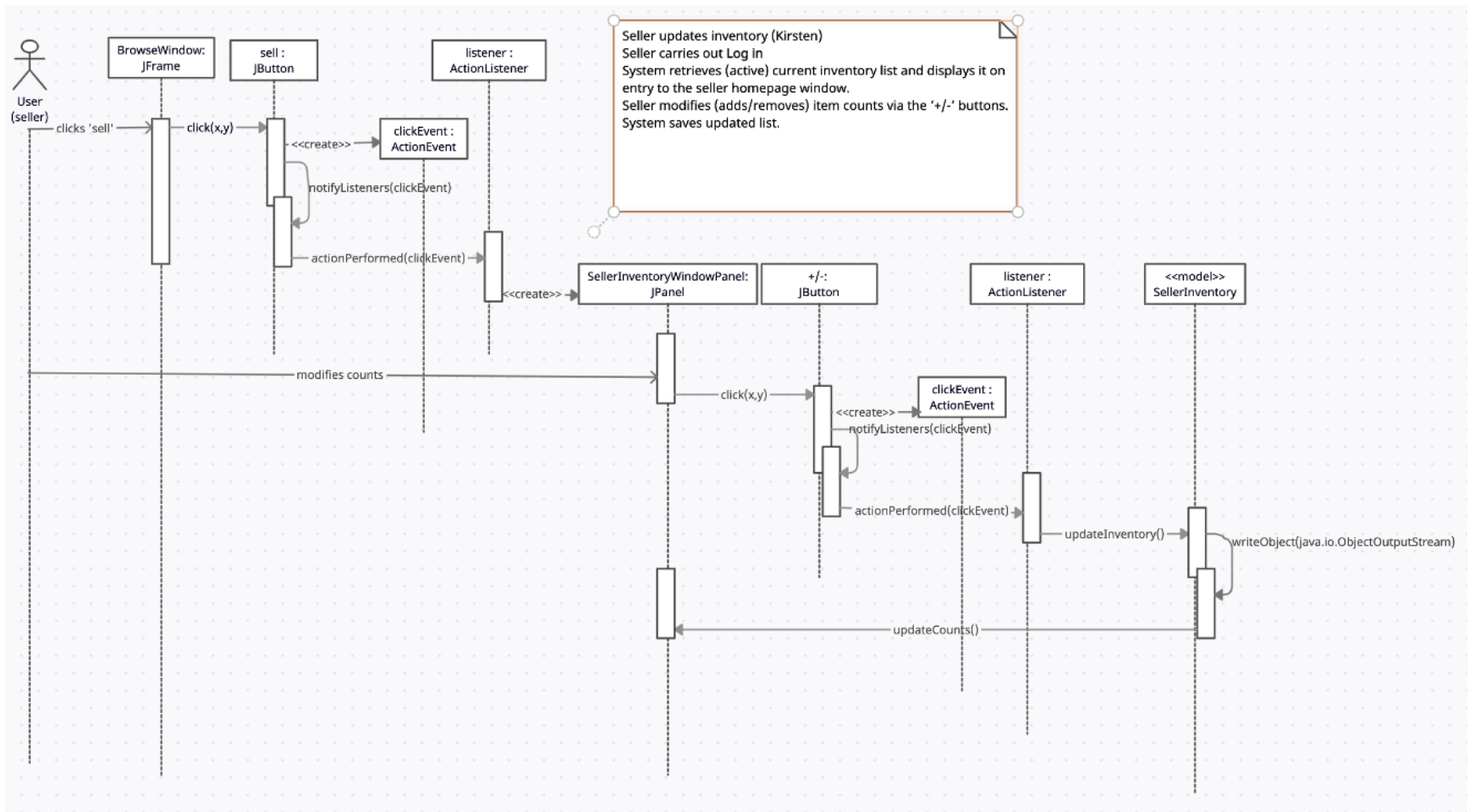
Strategy Pattern Implemented here

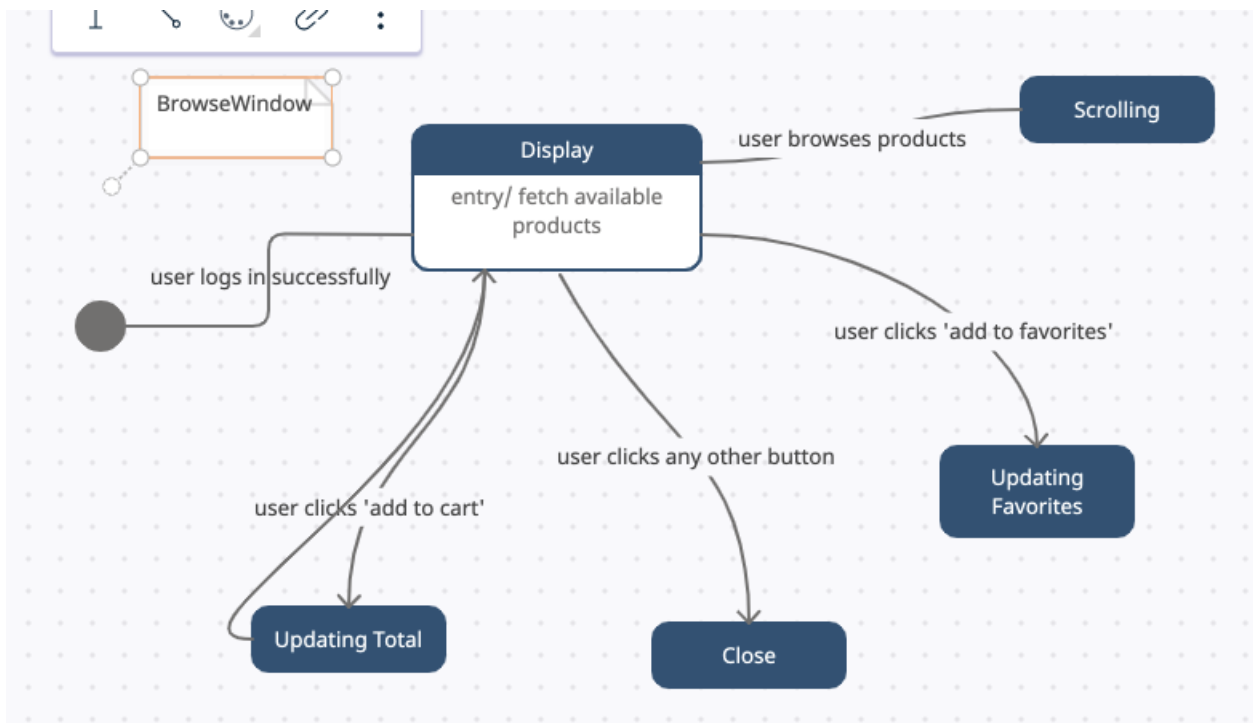
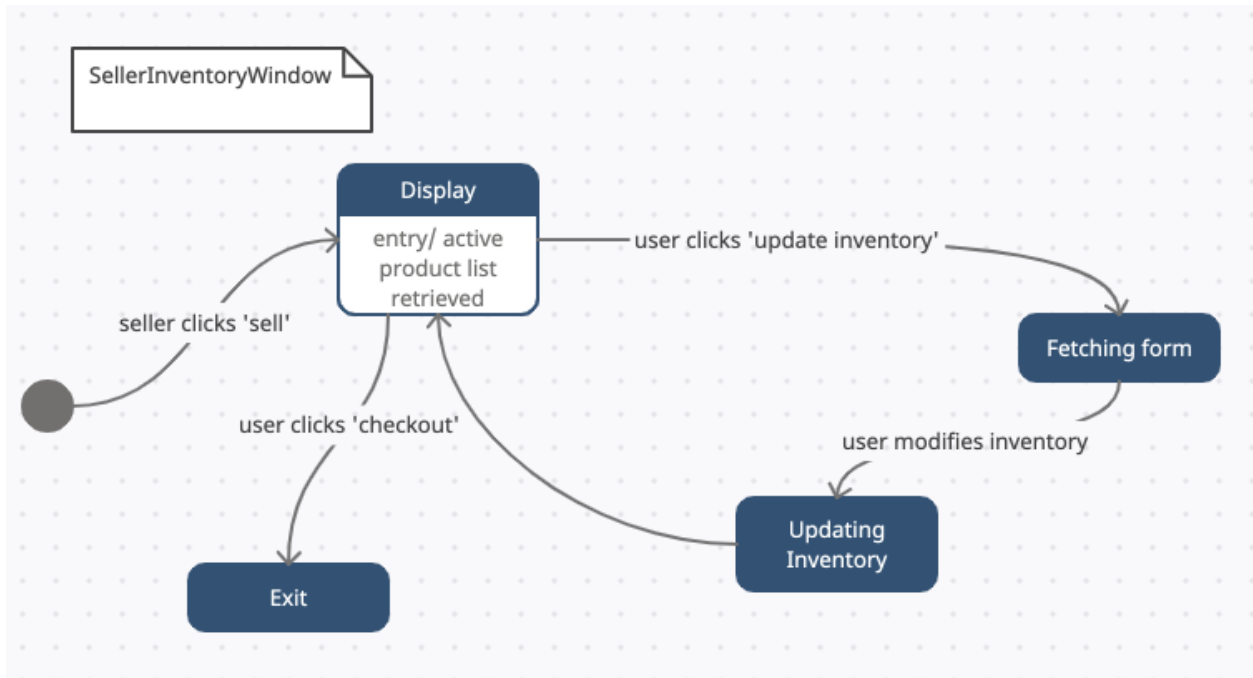
Buyer views order status/details  
 Buyer carries out Log in  
 Buyer clicks 'view orders' button  
 System retrieves list of Buyer's past orders  
 App displays order details (i.e, products ordered, quantity ordered, price)

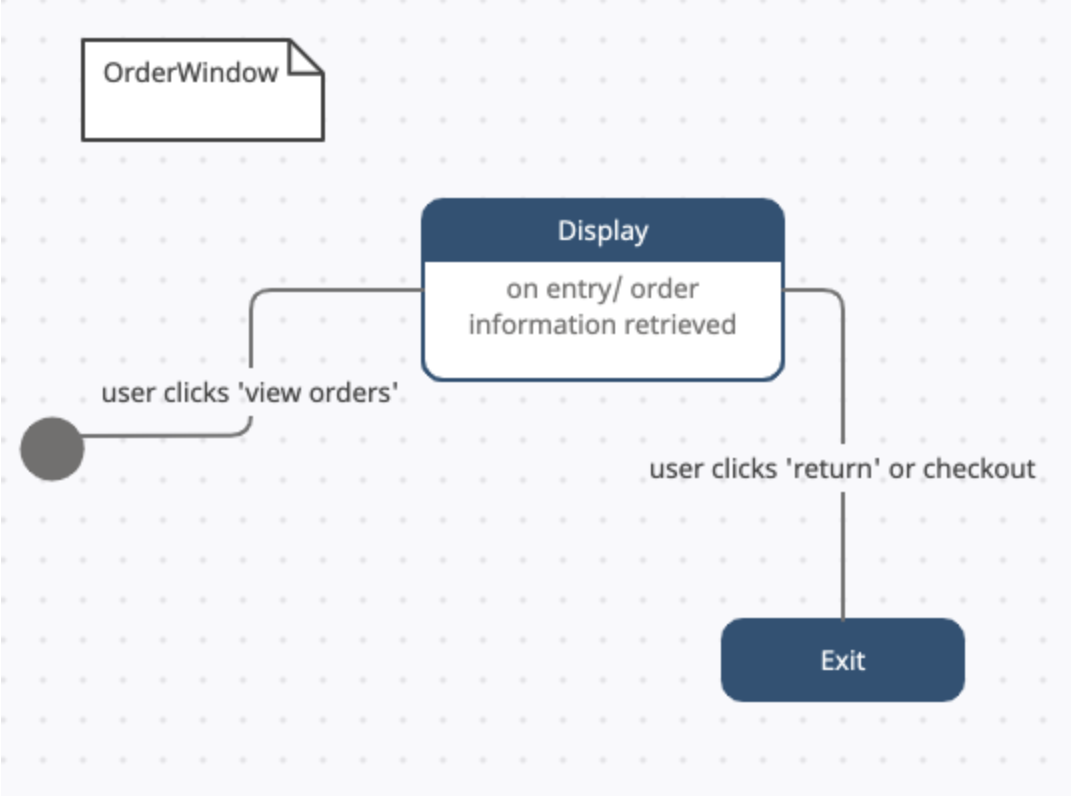




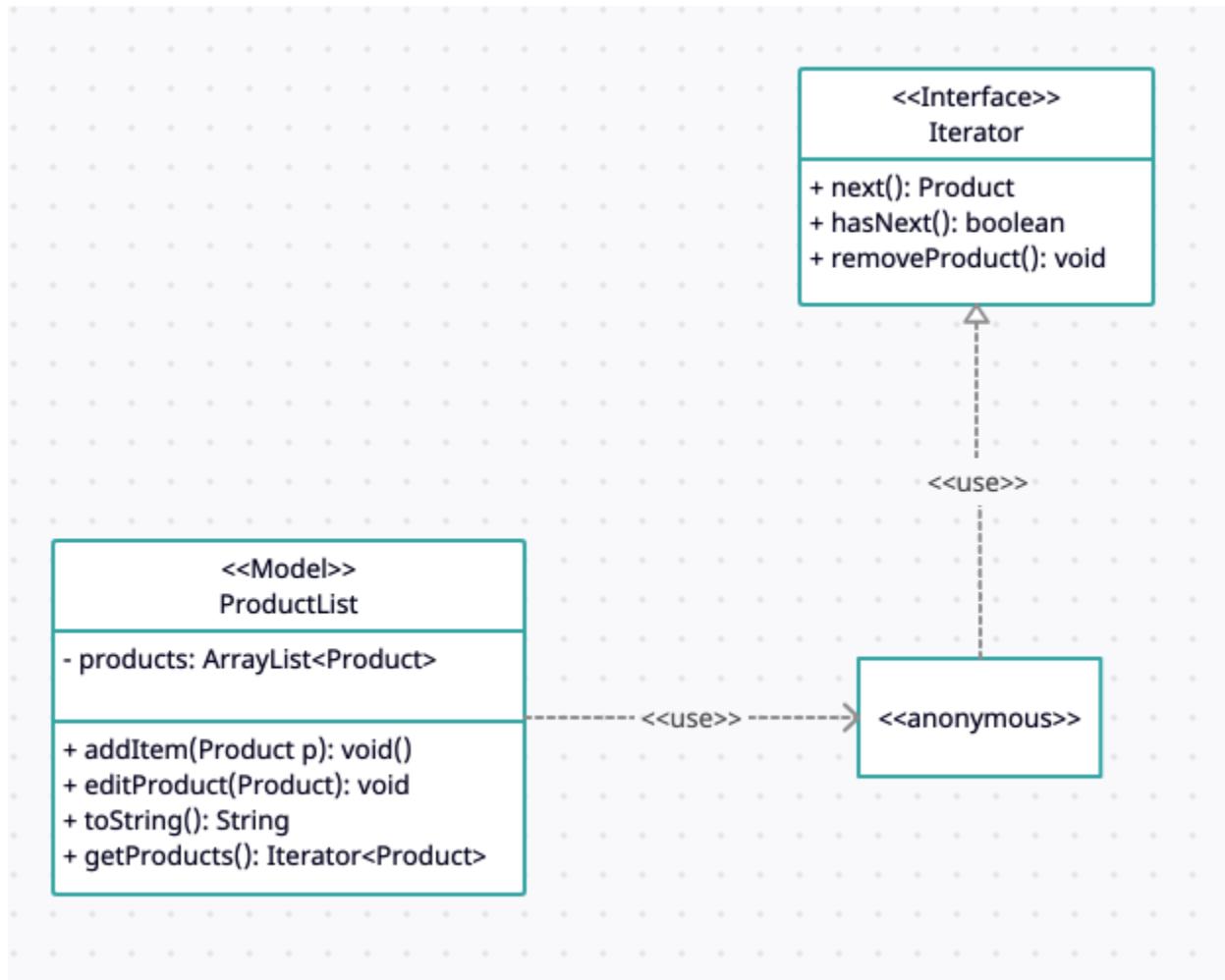




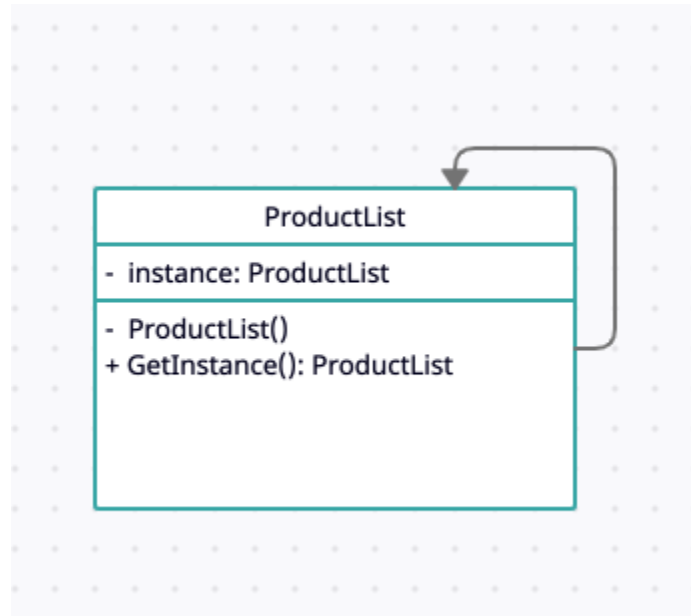




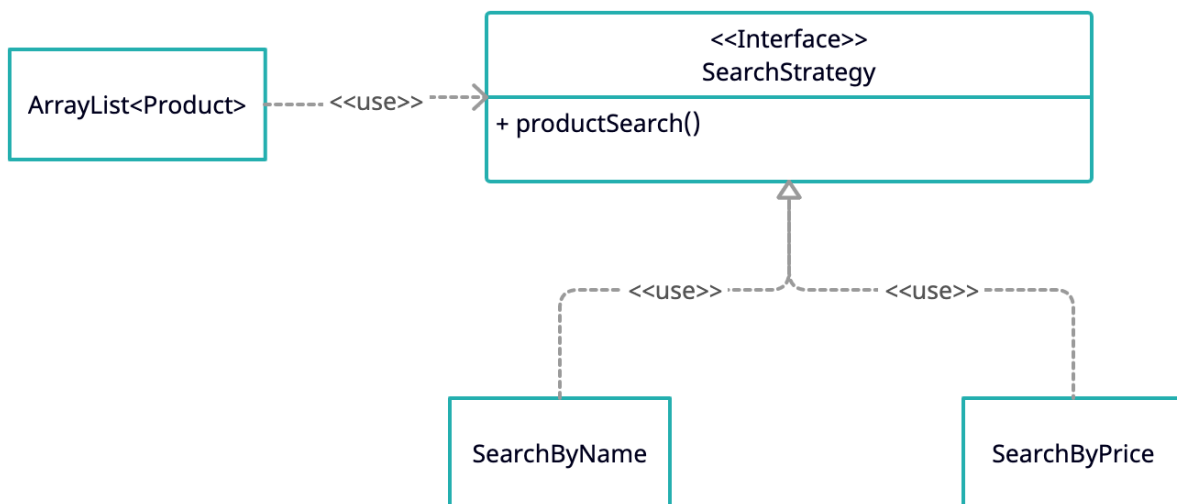
## Iterator Pattern



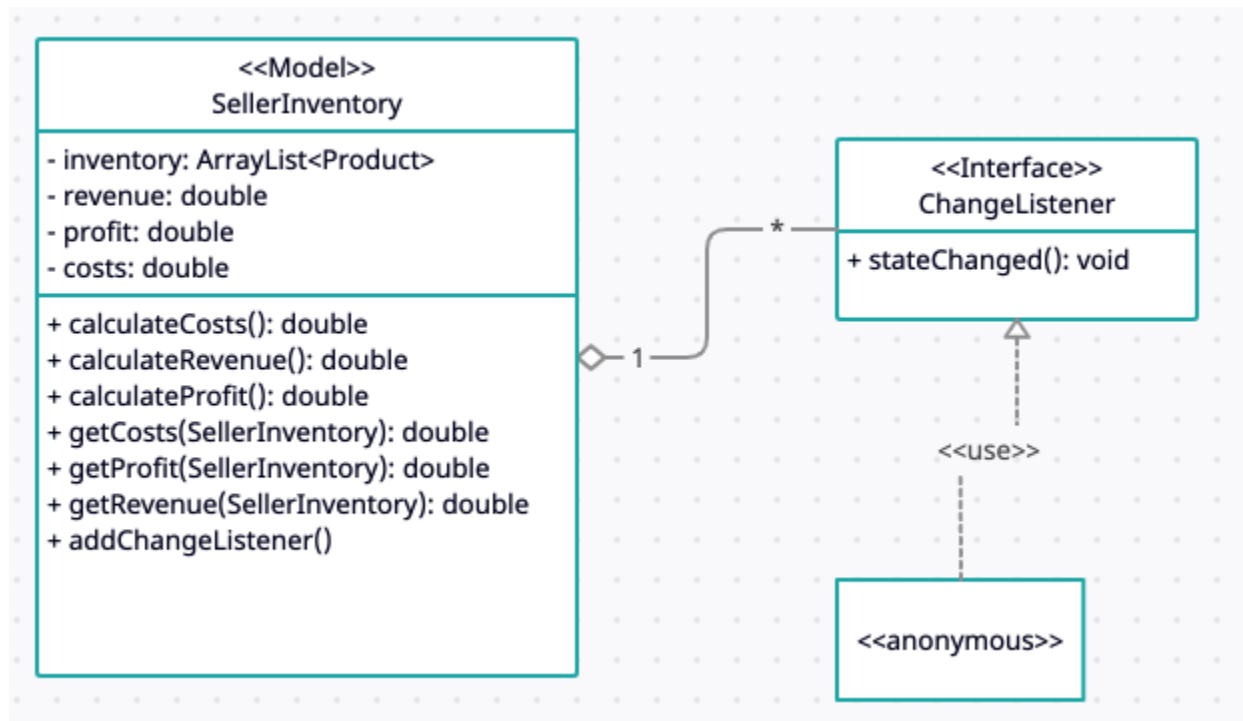
## Singleton Pattern



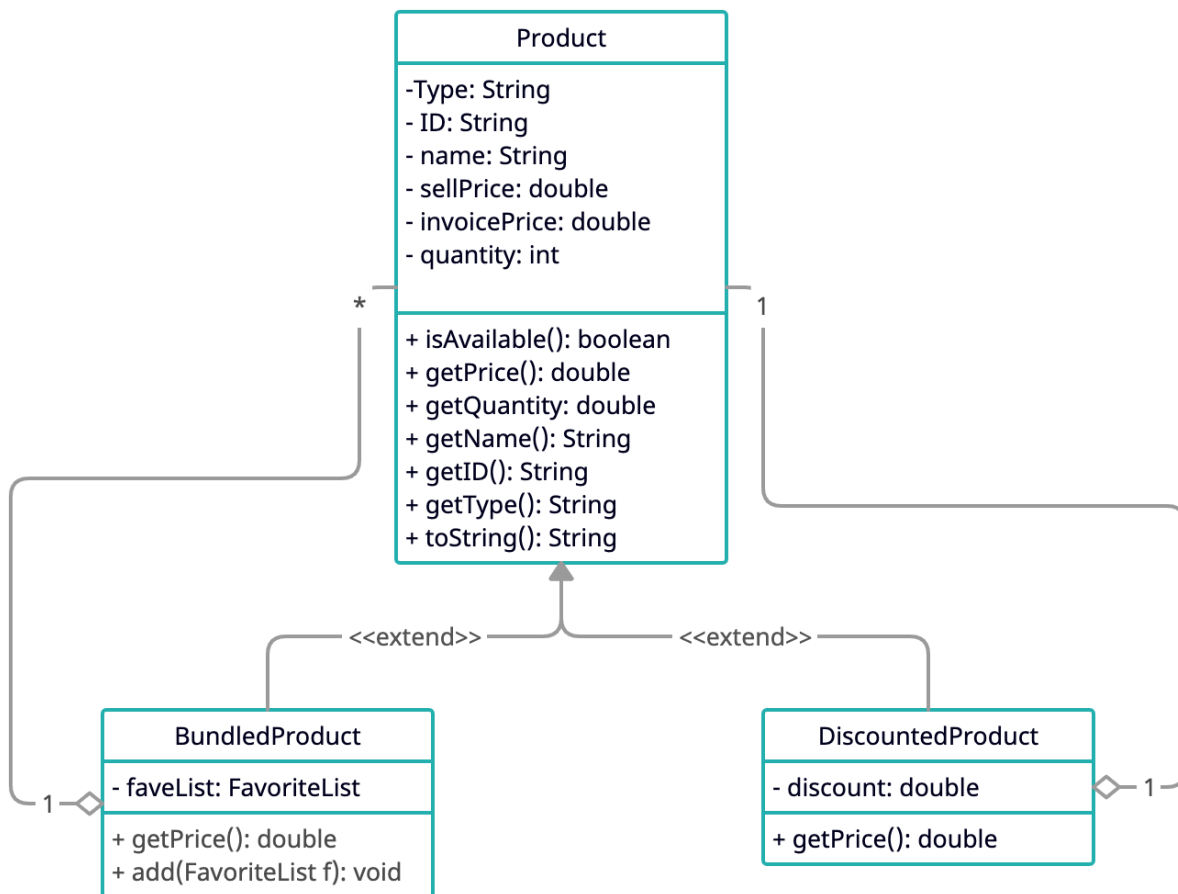
## Strategy Pattern



## Observer Pattern



## Decorator & Composite Pattern



## Project Code

### UserDB.java

```
package client;
```

```
import java.util.ArrayList;
```

```
/**  
 * @author JamesJenson  
 * a class that constructs and populates a user database.  
 */
```

```
public class UserDB {
```

```
    /**  
     * constructs a user database with buyers and sellers.  
     */
```

```
    public UserDB() {  
        sellers=new ArrayList<Seller>();  
        buyers=new ArrayList<Buyer>();  
        sellers.add(new Seller("1s","abcde","12345"));  
        sellers.add(new Seller("2s","RonaldMcDonald","lovinit"));  
        buyers.add(new Buyer("3b","randomdave","12345"));  
        buyers.add(new Buyer("4b","dave2.0","arealpassword2.0"));  
    }
```

```
    /**  
     * verifies the login information of a buyer and seller.  
     * @param userIn the username of a user  
     * @param passIn the password of a user  
     * @return the id number of a user  
     */
```

```
    public String verifyLogin(String userIn, String passIn) {  
        for (Seller s:sellers) {  
            if(s.getUsername().equals(userIn)&&s.checkPassword(passIn)) {  
                return s.getId();  
            }  
        }  
        for (Buyer b:buyers) {  
            if(b.getUsername().equals(userIn)&&b.checkPassword(passIn)) {  
                return b.getId();  
            }  
        }  
        return null;  
    }
```

```
    private ArrayList<Seller> sellers; //need to return some value that differentiates between buyers and sellers AND  
    returns the userid
```



```
private ArrayList<Buyer> buyers;
}
```

## **Buyer.java**

```
package client;
import java.io.Serializable;
import java.util.ArrayList;
/**
 * @author James Jenson
 * a class that creates a user of type buyer.
 */
public class Buyer implements Serializable{

    /**
     * creates a new buyer
     * @param idIn , the id number of the buyer.
     * @param userIn , the username of the buyer.
     * @param passwordIn , the password of the buyer.
     */
    public Buyer(String idIn, String userIn, String passwordIn) {
        id=idIn;
        username=userIn;
        password=passwordIn;
    }

    /**
     * retrieves the id of the buyer.
     * @return id, the id of the buyer.
     */
    public String getId() {
        return id;
    }

    /**
     * gets the buyers username.
     * @return username, the username of the buyer.
     */
    public String getUsername() {
        return username;
    }

    /**
     * validates the password of the buyer.
     * @param input , the string to verify against the password.
     * @return true or false, depending on validity of password.
     */
}
```

```

    */
    public boolean checkPassword(String input) {
        return (password.equals(input));
    }

    private String id;
    private String username;
    private String password;
    private ShoppingCart myCart;
    private ArrayList<Product> myFavorites;
}

```

## **Seller.java**

```

package client;

/**
 * @author James Jenson
 * a class that constructs a seller.
 */
public class Seller {
    public Seller(String idIn, String userIn, String passwordIn) {
        id=idIn;
        username=userIn;
        password=passwordIn;
    }

    /**
     * gets the seller's id.
     * @return the seller's id.
     */
    public String getId() {
        return id;
    }

    /**
     * gets the seller's username.
     * @return username, the seller's username.
     */
    public String getUsername() {
        return username;
    }

    /**
     * validates the password of the seller.
     * @param input the string to verify.

```

```

    * @return true or false, depending on whether the input was correct.
    */
    public boolean checkPassword(String input) {
        System.out.println(input+password);
        return (password.equals(input));
    }

    private String id;
    private String username;
    private String password;
}

```

## ShoppingCart.java

```

package client;

import java.io.*;
import java.util.ArrayList;
import java.util.Collections;

/**
 * @author James Jenson, Jared Usher, Kirsten Hernquist
 * a class that constructs an instance of and manipulates a shopping cart object.
 */
public class ShoppingCart implements Serializable{

    /**
     * constructs an instance of a shopping cart object.
     * @param userID the id of the buyer.
     * @throws IOException
     * @throws ClassNotFoundException
     */
    public ShoppingCart(String userID) throws IOException, ClassNotFoundException {
        this.userid = userID;
    }

    /**
     * get the quantity of the product in the cart.
     * @precondition: the available quantity of the product > 0.
     * @param p the product to retrieve the quantity of.
     * @return count, the number of occurrences the product is in the cart.
     * @postcondition: the cart quantity is updated accordingly.
     */
    public int getQuantity(Product p){

```

```

    int count = 0;
    for (Product i : cartContents)
    {
        if (i.getProductID().equals(p.getProductID()))
        {
            count++;
        }
    }
    return count;
}

/**
 * calculates the total cost of all items in the cart.
 */
public void calculateTotalPrice(){
    totalPrice = 0;
    cartContents.forEach(Product-> this.totalPrice = totalPrice + Product.getSellPrice());
}

/**
 * retrieves the total price of the cart contents.
 * @return the total price of the cart.
 */
public double getTotalPrice(){
    return totalPrice;
}

/**
 * adds an item to the cart then sorts the cart by name to group the
 * same items together within the list
 * @param p the product to add to the cart.
 * @throws IOException
 */
public void addItem(Product p) throws IOException {
    cartContents.add(p);
    Collections.sort(cartContents, sortByName.productSearch());
    saveCart();
}

/**
 * gets the contents of the cart.
 * @return the contents of the cart.
 * @precondition: the file exists and has been saved to previously.
 * @throws IOException
 * @throws ClassNotFoundException
 */
public ArrayList<Product> getCartContents() throws IOException, ClassNotFoundException {
    try {
        String fileName=userid+"cart.txt";

```

```

File file=new File(fileName);
if(file.exists()) {
    FileInputStream fis = new FileInputStream(fileName);//fileName);
    ObjectInputStream ois = new ObjectInputStream(fis);
    this.cartContents = (ArrayList<Product>) ois.readObject();
}
return cartContents;
}
catch(EOFException e)
{
    System.out.println("Cart is empty.");
}
return null;
}

```

```

/**
 * serializes the cart contents.
 * @postcondition: the file exists and has been serialized to.
 * @throws IOException
 */

```

```

public void saveCart() throws IOException {
    String fileName = userid + "cart.txt";
    File file=new File(fileName);
    if(file.exists())
    {
        FileOutputStream fos = new FileOutputStream(fileName);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(this.cartContents);
        oos.flush();
        oos.close();
        fos.close();
    }
    else{
        file.createNewFile();
    }
}

```

```

/**
 * removes an item from the cart.
 * @param p the products to remove from the cart.
 * @precondition: cart size > 0
 * @throws IOException
 */

```

```

public void removeItem(Product p) throws IOException {
    for(Product l: this.cartContents)
        System.out.println(l.getProductName());
    this.cartContents.remove(p);
    this.saveCart();
}

```

```

/**
 * removes all items from the cart.
 * @orecondition: the file exists.
 * @throws IOException
 */
public void emptyCart() throws IOException {
    String fileName = userid + "cart.txt";
    File file = new File(fileName);
    if (file.exists())
    {
        cartContents = new ArrayList<>();
        totalPrice = 0.00;
        this.saveCart();
    }
}

private double totalPrice;
SortByName sortByName = new SortByName();
private String userid;
private ArrayList<Product> cartContents = new ArrayList<>();
}

```

## **Product.java**

```
package client;
```

```
import java.io.Serializable;
```

```

/**
 * @author James Jenson
 * The Product class stores data on a single product. Name, seller ID, quantity, etc.
 */
public class Product implements Serializable {
    private String productDescription;
    private String productName;
    private int availableQuantity;
    private String productId;
    private double sellPrice;
    private double invoicePrice;
    private String type;
    private String sellerID;
    private int quantitySold;
    private int quantityInvoiced;
}

```

```

public Product() {};

/**
 * Product constructor.
 * @param name Name of new Product
 * @param productID ID of Product
 * @param prodType Product category of new Product. Unused right now I think other than display in popup
 * @param salePrice the sale price
 * @param desc A description of the product
 * @param available the starting quantity of items available
 * @param invPrice invoice price
 * @param sellerId the seller's ID
 * @postcondition new Product has been constructed
 * */

public Product(String name, String productID, String prodType, double salePrice, String desc, int available,
double invPrice, String sellerId) {
    this.availableQuantity = available;
    this.productDescription = desc;
    this.productName = name;
    this.productId = productID;
    this.sellPrice = salePrice;
    this.type = prodType;
    this.invoicePrice = invPrice;
    this.sellerID = sellerId;
    quantitySold=0;
    quantityInvoiced=available;
}

/**
 * Returns the current quantity of products available
 * @return quantity
 *
 * */

public int getAvailableQuantity() {
    return this.availableQuantity;
}

/**Returns product name
 * @return name
 * */

public String getProductName() {
    return this.productName;
}

/**Sets quantity available. Also changes the amount invoiced by the amount changed.
 * @precondition quantity >= 0
 * @param quantity sets quantityAvailable to this number
 * @postcondition availableQuantity is set to quantity, invoiced quantity has been changed by correct amount.
 * */

```

```

public void setAvailableQuantity(int quantity) {
    if(quantity<0)return;
    quantityInvoiced+=(quantity-availableQuantity);
    this.availableQuantity = quantity;
}

/**Returns product ID
 * @return int product ID
 * */
public String getProductID() {
    return this.productID;
}

/**Returns user ID of seller of product
 * @return seller ID
 * */
public String getSellerID() {
    return this.sellerID;
}

/**Returns product invoice price
 * @return invoice price
 * */
public Double getInvoicePrice() {
    return this.invoicePrice;
}

/**Returns product selling price
 * @return sellPrice
 * */
public double getSellPrice() {
    return this.sellPrice;
}

/**Returns product name
 * @return name
 * */
public String getType() {
    return this.type;
}

/**Returns Product description String
 * @return desc string
 * */
public String getProductDescription(){
    return this.productDescription;
}

/**Decreases number of available and increases sell count tracker

```



```

* @precondition amount is less than 0, amount is less than or equal to quantityAvailable
* @return true if successful, false otherwise
* */
public boolean sell(int amount) {
    if(amount<0) return false;
    if(availableQuantity<amount) return false;
    quantitySold+=amount;
    availableQuantity-=amount;
    return true;
}

/**Returns the quantity of this product that has been sold
* @return amount of this product sold
* */
public int getAmountSold() {
    return quantitySold;
};

/**Returns amount of this Product invoiced by the seller
* @return int quantity Invoiced
* */
public int getAmountInvoiced() {
    return quantityInvoiced;
}
}

```

## **DiscountedProduct.java**

```

package client;

import java.util.ArrayList;

/**
* @author Kirsten Hernquist
* /
public class DiscountedProduct extends Product{
    private String productName;
    private String productid;
    private String productType;
    private double sellPrice;
    private String description;
    private int quantityAvailable;
    private String sellerid;
    private double invoicePrice;

    /**

```

```

* constructs a discounted product.
* @param name the name of the product.
* @param productID the id of the product.
* @param prodType the type of the product.
* @param salePrice the sale price of the item.
* @param desc the description of the product.
* @param available the number of a given item that is available.
* @param invPrice the invoice price of the item.
* @param sellerId the sellerid associated with an item.
*/
public DiscountedProduct(String name, String productID, String prodType, double salePrice, String desc, int
available, double invPrice, String sellerId){//roduct p) {
    super(name, productID, prodType, salePrice, desc, available, invPrice, sellerId);
}

/**
* applies a discount to the product bundle.
* @return the total price minus the discount (10%).
*/
@Override
public double getSellPrice(){
    return (super.getSellPrice() * (1-discount));
}

/**
* gets the available quantity of the product.
* @return 1, the number of a unique bundle available.
*/
@Override
public int getAvailableQuantity() {
    return 1;
}

/**Returns product name
* @return int product name
* */
@Override
public String getProductName() {

    return super.getProductName();
}

/**Returns product ID
* @return int product ID
* */
@Override
public String getProductID(){
    return super.getProductID();
}

```

```
}
```

```
/**Returns seller ID  
 * @return string seller ID  
 */
```

```
@Override
```

```
public String getSellerID() {  
    return super.getSellerID();  
}
```

```
/**Returns product invoice price  
 * @return double product invoice price  
 */
```

```
@Override
```

```
public Double getInvoicePrice() {  
    return super.getInvoicePrice();  
}
```

```
/**  
 * assigns the product type as FavoritesBundle  
 * @return "FavoritesBundle"  
 */
```

```
@Override
```

```
public String getType() {  
    return "FavoritesBundle";  
}
```

```
/**  
 * gets the description of the product.  
 * @return "FavoritesBundle".  
 */
```

```
@Override
```

```
public String getProductDescription() {  
    return "FavoritesBundle";  
}
```

```
/**  
 * a function that tracks the number of items sold.  
 * @param amount the amount sold.  
 * @return true or false  
 */
```

```
@Override
```

```
public boolean sell(int amount) {  
    for (Product p : products)  
    {  
        if(amount<0) return false;  
        if(availableQuantity<amount) return false;  
        quantitySold+=amount;  
    }  
}
```

```

        availableQuantity-=amount;
    }
    return true;
}

/**Returns the quantity of this product that has been sold
 * @return amount of this product sold
 */
@Override
public int getAmountSold() {
    return 1;
};

/**Returns amount of this Product invoiced by the seller
 * @return int quantity Invoiced
 */
@Override
public int getAmountInvoiced() {
    return 1;
}

private ArrayList<Product> faveList = new ArrayList<>();
private String userid;
private ArrayList<Product> products = new ArrayList<>();
private int quantitySold;
private int availableQuantity;
private Product product;
private final double discount = 0.10;
}

```

## SingletonProductList.java

```

package client;

import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * @author Kirsten Hernquist, James Jenson
 * creates and manipulates a singleton object.
 */
public class SingletonProductList implements Serializable {

    /**

```

```

* gets the instance of the singleton.
* @return instance
* @throws IOException
* @throws ClassNotFoundException
*/
public static SingletonProductList getInstance() throws IOException, ClassNotFoundException {
    if (instance == null) {
        instance = new SingletonProductList();
    }
    return instance;
}

```

```

/**
 * ensures that the singleton design pattern is maintained throughout serialization.
 * @return the instance
 * @throws IOException
 * @throws ClassNotFoundException
 */
public Object readResolve() throws IOException, ClassNotFoundException {
    return getInstance();
}

```

```

/**
 * adds an item to the list
 * @param p the product to add
 * @throws IOException
 * @postcondition: the list is serialized.
 */
public void add(Product p) throws IOException {
    productList.add(p);
    this.saveList(this.productList);
}

```

```

/**
 * gets an item
 * @param index the index of the item to retrieve.
 * @precondition: the list != null
 */
public void getItem(int index) {
    productList.get(index); //get product at specific index
}

```

```

/**
 * updates the available quantity of a product in the list.
 * @param productID the id of the product to update.
 * @param count the number available to purchase.
 * @precondition: the product id is valid and the count > 0.
 * @throws IOException
 * @postcondition: serializes the updated list.

```

```

*/
public void updateAvailableQuantity(String productID, int count) throws IOException {
    if (count < 0)
    {
        //thrown new InvalidParameterException("Quantity must be greater than or equal to 0.");
    }
    for (Product p : productList) {

        if (productID.equals(p.getProductID())) {

            int currentQuantity = count;

            p.setAvailableQuantity(currentQuantity);
        }
    }
    this.saveList(this.productList);
}

```

```

/**
 * adds or subtracts item quantities one by one.
 * @param productID the id of the product to be added or subtracted from
 * @param count the indicator of subtraction or addition.
 * @postcondition: serializes the updated list.
 * @throws IOException
 */
public void addOrSubtractQuantity(String productID, int count) throws IOException {

    for (Product p : productList) {

        if (productID.equals(p.getProductID())) {

            int currentQuantity = p.getAvailableQuantity();

            if (count < 0) {
                if (currentQuantity > 0)
                {System.out.println("dec: " + currentQuantity--);
                 p.setAvailableQuantity(currentQuantity--);}

            } else {
                System.out.println("inc: " + currentQuantity++);
                p.setAvailableQuantity(currentQuantity++);
            }
        }
    }
    this.saveList(this.productList);
}

```

```

/**
 * dictates the number of a certain product to sell.

```

```

* @param productID the id of the product to sell.
* @param count the amount of the product to sell.
* @postcondition: serializes the updates list.
* @throws IOException
*/
public void sellQuantity(String productID, int count) throws IOException {
    for (Product p : productList) {
        if (productID.equals(p.getProductID())) {
            if (p.getAvailableQuantity() >= count) {
                p.sell(count);
            }
            else {
                p.sell(p.getAvailableQuantity());
            }
        }
    }
    this.saveList(this.productList);
}

/**
* serializes the product list.
* @param productList the list to serialize.
* @throws IOException
* @precondition: the file exists.
* @postcondition: the list is serialized.
*/
public void saveList(ArrayList<Product> productList) throws IOException {
    FileOutputStream fos = new FileOutputStream("globalproductlist.txt");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(this.productList);
    oos.flush();
    oos.close();
    fos.close();
}

/**
* creates an iterator for the list.
* @return Iterator, the iterator for the list.
*/
public Iterator<Product> getWholeList() {
    return new Iterator<Product>() {
        @Override
        public boolean hasNext() {
            return current < productList.size();
        }

        @Override
        public Product next() {
            return productList.get(current++);
        }
    };
}

```

```

    }

    private int current = 0;
};
}

/**
 * gets the product list for the browse window, which returns the WHOLE list independent of sellerid.
 * @return ArrayList, a list of all products on sale.
 */
public ArrayList<Product> getProductList(){
    try {
        FileInputStream fis = new FileInputStream("globalproductlist.txt");
        ObjectInputStream ois = new ObjectInputStream(fis);
        productList = (ArrayList<Product>) ois.readObject();
    }
    catch(IOException | ClassNotFoundException e)
    {
        System.out.println("There are no products for sale.");
    }
    return productList;
}

/**
 * gets a list used by sellerinventorywindow, which returns only the items that belong to that seller.
 * @param userID the id of the seller to whom the list belongs.
 * @return ArrayList of products, the seller's inventory.
 * @precondition: the file exists.
 * @throws IOException
 * @throws ClassNotFoundException
 */
public ArrayList<Product> getList(String userID) throws IOException, ClassNotFoundException {
    File file = new File("globalproductlist.txt");
    if (file.exists())
    {
        FileInputStream fis = new FileInputStream("globalproductlist.txt");
        ObjectInputStream ois = new ObjectInputStream(fis);
        productList = (ArrayList<Product>) ois.readObject();
        ois.close();

        ArrayList<Product> personalProductList = new ArrayList<>();
        for (Product p : productList)
            if (p.getSellerID().equals(userID)) {
                personalProductList.add(p);
            }
        ois.close();
        return personalProductList;
    }
}

```



```

        return new ArrayList<Product>();
    }

    private static SingletonProductList instance = null;
    private ArrayList<Product> productList = new ArrayList<>();
    private ArrayList<Product> pList = new ArrayList<>();
    private static final long serialVersionUID = 6529685098267757690L;
}

```

## Order.java

```

package client;

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

/**
 * The Order Class Stores Data On Several Products And Total Price Of Them
 * @author Jared Usher
 */
public class Order implements Serializable {

    /**
     * Order Constructor
     * @param sC Shopping Cart To Be Converted Into an Order
     * @postcondition new Order has been constructed
     */
    public Order(ShoppingCart sC)
    {
        try
        {
            sC.calculateTotalPrice();
            this.orderProducts = sC.getCartContents();
            this.orderPrice = sC.getTotalPrice();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Get Product Contents Of An Order

```

```

* @return Products In The Order
* @throws IOException
* @throws ClassNotFoundException
*/
public ArrayList<Product> getOrderProducts() throws IOException, ClassNotFoundException
{
    return this.orderProducts;
}

/**
* Get Total Cost Of An Order
* @return double Order Price
*/
public double getOrderPrice()
{
    return this.orderPrice;
}

/**
* Compares Two Orders
* An Order Must Be Exactly Identical (Including Order Of Products)
* Only Used For Testing Purposes
* @param o Order To Be Compared With
* @return True If Equals, False Otherwise
*/
public boolean equals(Order o)
{
    if (this.orderPrice == o.getOrderPrice())
    {
        for (int i = 0; i < this.orderProducts.size(); i++)
        {
            try
            {
                String thisID = this.orderProducts.get(i).getProductID();
                String otherID = o.getOrderProducts().get(i).getProductID();

                if (!thisID.equals(otherID))
                {
                    return false;
                }
            }
            catch (Exception e) {}
        }

        return true;
    }
    else
    {

```

```

        return false;
    }
}

private ArrayList<Product> orderProducts = new ArrayList<>();
private double orderPrice;
}

```

## OrderList.java

```

package client;

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

/**
 * The Order Class Stores All Orders Of A User
 * @author Jared Usher
 */
public class OrderList
{
    /**
     * OrderList Constructor
     * @param userID The Buyer's ID
     */
    public OrderList(String userID){
        this.userid = userID;
    }

    /**
     * Adds A New Order To A List Of The User's Previous Orders,
     * Serializes List After Adding Order
     * @param o Order To Be Added
     * @throws IOException
     */
    public void addOrder(Order o) throws IOException {
        try {this.getOrders();} catch(Exception e) {}
        orders.add(o);
        this.saveOrderList();
    }

    /**
     * Saves The List Of A User's Past Orders To A File
     * @throws IOException
     */
    public void saveOrderList() throws IOException {

```

```

String fileName = userid + "Orders" + ".txt";
FileOutputStream fos = new FileOutputStream(fileName);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(this.orders);
oos.flush();
oos.close();
fos.close();
}

/**
 * Loads The List Of A User's Past Orders From File
 * @return orders Containing List Of All Of A User's Past Orders
 * @throws IOException
 * @throws ClassNotFoundException
 */
public ArrayList<Order> getOrders() throws IOException, ClassNotFoundException {
    String fileName = userid + "Orders" + ".txt";
    File file=new File(fileName);
    if(file.exists())
    {
        FileInputStream fis = new FileInputStream(fileName);
        ObjectInputStream ois = new ObjectInputStream(fis);
        this.orders = (ArrayList<Order>) ois.readObject();
    }
    return orders;
}

private String userid;
private ArrayList<Order> orders = new ArrayList<>();
}

```

## **FavoritesList.java**

```

package client;

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

/**
 * @author Kirsten Hernquist
 * a class that constructs and manipulates a favorites list of a user.
 */
public class FavoritesList implements Serializable {

    /**
     * constructs a favorites list

```

```
* @param userID, the userid of whom to create a favorite list for.  
*/
```

```
public FavoritesList(String userID){  
    this.userid = userID;  
}
```

```
/**
```

```
* adds a product to the favorites list.  
* @param p the product to add to the favorites list.  
* @throws IOException  
*/
```

```
public void addFave(Product p) throws IOException {  
    try {  
        this.getFaves();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
    for (Product i : faves){  
        if (p.getProductID().equals(i.getProductID()))  
        {  
            System.out.println("Product already in favorites");  
            return;  
        }  
    }  
}
```

```
    faves.add(p);  
    this.saveFaveList();  
}
```

```
/**
```

```
* removes a product from the favorites list.  
* @precondition favoriteslist size > 0  
* @param p the product to remove  
* @throws IOException  
*/
```

```
public void removeFave(Product p) throws IOException {  
    try {  
        this.getFaves();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    if (faves.size() == 0)  
    {  
        System.out.println("Favorites list is empty.");  
        return;  
    }  
    for (int i = 0; i < faves.size(); i++)  
    {
```

```

        if (p.getProductID().equals(faves.get(i).getProductID()))
        {
            System.out.println(p.getProductName() + " removed from favorites");
            faves.remove(i);
            this.saveFaveList();
            return;
        }
    }
}

/**
 * serializes the favoriteslist
 * @throws IOException
 */
public void saveFaveList() throws IOException {
    String fileName = userid + ".txt";
    File file = new File(fileName);
    if (file.exists())
    {
        FileOutputStream fos = new FileOutputStream(fileName); //fileName);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(this.faves);
        oos.flush();
        oos.close();
        fos.close();
    }
    else
    {
        file.createNewFile();
    }
}

/**
 * deserializes the favoriteslist.
 * @return faves, the users favoriteslist
 * @throws IOException
 * @throws ClassNotFoundException
 * @precondition: the file exists.
 */
public ArrayList<Product> getFaves() throws IOException, ClassNotFoundException {
    try{
        String fileName=userid+".txt";
        File file=new File(fileName);
        if(file.exists()) {

            FileInputStream fis = new FileInputStream(fileName);
            ObjectInputStream ois = new ObjectInputStream(fis);

```

```

        this.faves = (ArrayList<Product>) ois.readObject();
    }}
    catch (EOFException e)
    {
        System.out.println("No products in favorites list.");
    }
    return faves;
}

private String userid;
private ArrayList<Product> faves = new ArrayList<>();
}

```

## FavoritesBundle.java

```
package client;
```

```
import java.io.IOException;
import java.util.ArrayList;
```

```
/**
 * @author Kirsten Hernquist
 * constructs and manipulates a bundled product from the buyer's favorites list.
 */
```

```
public class FavoritesBundle extends Product{
```

```

    /**
     * constructs a bundled product using the buyer's favorites list.
     * @param userID the id of the user to get the favorites of.
     * @throws IOException
     * @throws ClassNotFoundException
     * @precondition the favoriteslist is not empty.
     */
    public FavoritesBundle(String userID) throws IOException, ClassNotFoundException {
        this.userid = userID;
        FavoritesList faves = new FavoritesList(userID);
    }

```

```

    /**
     * adds a product to the bundle.
     * @param p the product to add.
     */
    public void add(Product p)
    {
        products.add(p);
    }

```

```

/**
 * calculates the sale price of all aggregated items as one.
 * @return price, the total price of the bundle.
 * @precondition the favoriteslist is not empty.
 * @postcondition the price is returned as a double.
 */

```

```

@Override
public double getSellPrice()
{
    double price = 0;
    for(Product p : products)
    {
        price += p.getSellPrice();
    }
    return price;
}

```

```

/**
 * returns the number of one unique discounted products available.
 * @return 1, the number of that bundle that exist.
 */

```

```

@Override
public int getAvailableQuantity() {
    return 1;
}

```

```

/**
 * creates the name of the product bundle.
 * @return bundleName, the aggregated name of the products.
 * @precondition bundle size > 0.
 */

```

```

@Override
public String getProductNames() {
    String bundleName = "";
    for(Product p : this.products)
    {
        bundleName = bundleName + " " + p.getProductNames() + " ";
    }
    return bundleName;
}

```

```

/**
 * gets the id number of the product

```



```

* @return id, the concatenated id of all products in the bundle.
*/
@Override
public String getProductID(){
    String id = "";
    for (Product p : products)
    {
        id = id + " " + p.getProductID() + " ";
    }
    return id;
}

/**
* creates the sellerId for the bundle.
* @return sellerID, the aggregated sellerIDs of the products in the bundle.
*/
@Override
public String getSellerID() {
    String sellerID = "";
    for (Product p : products)
    {
        sellerID = sellerID + " " + p.getSellerID() + " ";
    }
    return sellerID;
}

/**
* calculates the invoice price of the bundle.
* @return invPrice, the summed prices of all products in the bundle.
*/
@Override
public Double getInvoicePrice() {
    double invPrice = 0;
    for (Product p : products)
    {
        invPrice = invPrice + p.getInvoicePrice();
    }
    return invPrice;
}

/**
* gets the type of the product.
* @return "FavoritesBundle", the product type.
*/
@Override
public String getType() {
    return "FavoritesBundle";
}

```

```

/**
 * gets the description of the product
 * @return "FavoritesBundle", the product description.
 */
@Override
public String getProductDescription(){
    return "FavoritesBundle";
}

/**Decreases number of available and increases sell count tracker
 * @precondition amount is less than 0, amount is less than or equal to quantityAvailable
 * @return true if successful, false otherwise
 */
@Override
public boolean sell(int amount) {
    for (Product p : products)
    {
        p.sell(amount);
    }
    return true;
}

/**Returns the quantity of this product that has been sold
 * @return amount of this product sold
 */
@Override
public int getAmountSold() {
    return 1;
};

/**Returns amount of this Product invoiced by the seller
 * @return int quantity Invoiced
 */
@Override
public int getAmountInvoiced() {
    return 1;
}

private ArrayList<Product> faveList = new ArrayList<>();
private String userid;
private ArrayList<Product> products = new ArrayList<>();
private int quantitySold;
private int availableQuantity;
}

```

## **SellerInventory.java**

```
package client;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;
```

```
import javax.swing.event.ChangeEvent;
```

```
/**
 * @author Kirsten Hernquist, James Jenson
 * a class that constructs and manipulates a seller's inventory.
 */
```

```
public class SellerInventory implements Serializable {
```

```
/**
 * constructs a seller's inventory and calculates the financials of the seller.
 * @param mySellerID the id of the seller
 * @throws IOException
 * @throws ClassNotFoundException
 */
```

```
public SellerInventory(String mySellerID) throws IOException, ClassNotFoundException {
    this.sellerID = mySellerID;
    this.singleton = SingletonProductList.getInstance();
    this.inventory = singleton.getList(sellerID);
    listeners = new ArrayList<>();
    this.calculateCosts();
    this.calculateRevenue();
    this.calculateProfits();
}
```

```
/**
 * calculates the revenue of the seller.
 */
```

```
public void calculateRevenue() {
    revenue=0;

    inventory.forEach(Product -> {
        System.out.println("sold:"+Product.getAmountSold());
        this.revenue = revenue + (Product.getSellPrice()*Product.getAmountSold());
    });

    calculateProfits();
}
```

```

}

/**
 * calculates the costs incurred by the seller.
 */
public void calculateCosts() {

    costs=0;
    inventory.forEach(Product->this.costs = costs + (Product.getInvoicePrice()*Product.getAmountInvoiced()));

    ChangeEvent event = new ChangeEvent(this);

    for(SellerFinancialView n : listeners)
    {
        n.stateChanged(event);
    }
    calculateProfits();
}

/** calculates the profits earned by the seller.
 *
 */
public void calculateProfits() {
    this.profit = this.revenue - this.costs;
}

/**
 * gets the costs incurred by the seller.
 * @return costs
 */
public double getCosts() {
    return costs;
}

/**
 * gets the profits made by the seller.
 * @return profit
 */
public double getProfit() {
    return profit;
}

/**
 * gets the revenue made by the seller.
 * @return revenue
 */
public double getRevenue() {
    return revenue;
}

```

```

/**
 * gets the inventory of the seller
 * @return ArrayList of the inventory
 * @throws IOException
 * @throws ClassNotFoundException
 */
public ArrayList<Product> getInventory() throws IOException, ClassNotFoundException { //or return
sellerInventory object?
    SingletonProductList pList = SingletonProductList.getInstance();
    ArrayList<Product> myStuff = new ArrayList<>();
    myStuff = pList.getList(sellerID);
    return myStuff;
}

/**
 * updates the quantity of an item in the seller's inventory
 * @param productID the id of the product to locate
 * @param count the quantity to update the item as having
 * @precondition: the product must exist in the inventory.
 * @postcondition: the available quantity of the product is updated accordingly.
 */
public void updateQuantity(String productID, int count){

    for (Product p : this.inventory)
    {
        if ( productID == p.getProductID())
        {
            int currentQuantity = p.getAvailableQuantity();
            if (count < 0)
            {
                p.setAvailableQuantity(currentQuantity--);
                System.out.println(p.getAvailableQuantity());
            }

            else{
                p.setAvailableQuantity(currentQuantity++);
            }
        }
    }
}

/**
 * adds a changelister to the list.
 * @param sfv the sellerfinancialview to add as a listener.
 */
public void addChangeListener(SellerFinancialView sfv) { //SellerFinancialView sfv (add jpanel)
    listeners.add(sfv);
}

```

```

private ArrayList<Product> inventory = new ArrayList<>();
private double revenue = 0;
private double profit = 0;
private double costs = 0;
private String sellerID;
private SingletonProductList singleton;
private ArrayList<SellerFinancialView> listeners;
private ArrayList<Product> soldItems = new ArrayList<>();

}

```

## SortStrategy.java

```

package client;

import java.util.Comparator;

/**
 * @author Kirsten Hernquist
 * an interface for sorting products.
 */
public interface SortStrategy {
    public Comparator<Product> productSearch();
}

```

## SortByName.java

```

package client;

import java.util.Comparator;

/**
 * @author Kirsten Hernquist
 * a class to sort products by name in alphabetical order.
 */
public class SortByName implements SortStrategy {
    /**
     * compares the products by name
     * @return nameSort, a comparator object
     */
    @Override
    public Comparator<Product> productSearch() {
        Comparator<Product> nameSort= new Comparator<Product>()

```

```

    {
        @Override
        public int compare(Product o1, Product o2) {
            return o1.getProductName().compareTo(o2.getProductName());
        }
    };
    return nameSort;
}
}

```

## SortByPrice.java

```
package client;
```

```
import java.util.Comparator;
```

```

/**
 * @author Kirsten Hernquist
 * a class that implements the sorting of products by price.
 */

```

```
public class SortByPrice implements SortStrategy {
```

```

    /**
     * sorts the products by price (low to high)
     * @return byPrice, a comparator object
     */

```

```
    @Override
```

```
    public Comparator<Product> productSearch() {
```

```
        Comparator<Product> byPrice = new Comparator<Product>() {
```

```

            public int compare(Product o1, Product o2) {
                int flag = 0;
                flag = Double.compare(o1.getSellPrice(), o2.getSellPrice());
                return flag;
            }

```

```
        };

```

```
        return byPrice;
```

```
    }
}

```

## BrowseWindow.java

```

package gui;

import java.util.Collections;
import java.util.Iterator;
import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.*;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import client.Order;
import client.Product;
import client.ShoppingCart;
import client.SortByName;
import client.SingletonProductList;
import client.SortByPrice;
import client.FavoritesList;

/**
 * @author Jared Usher
 * constructs a displays a browse window for the buyer to view what is on sale. Allows for adding to favorites, adding to
 * cart,
 * displays cart total, displays an option to view favorites, and allows for unfavoriting.
 */
public class BrowseWindow
{
    public BrowseWindow(String userID) throws IOException, ClassNotFoundException {
        this.userid = userID;
        this.myFavorites = new FavoritesList(userID);
        this.myCart = new ShoppingCart(userID);
        this.myFaveList = myFavorites.getFaves();
        this.myCartContents = myCart.getCartContents();

        /*for(Product p : myCartContents)
        {
            System.out.println("cart: " + p.getProductName());
        }*/
        myCart.calculateTotalPrice();
        //System.out.println("tot price: " + myCart.getTotalPrice());

        final JFrame browseWindow = new JFrame("Browse Products");
        browseWindow.setBounds(100, 100, 800, 550);
        browseWindow.setLayout(null);

        //View favorites button
        JButton viewFaveButton = new JButton("View Favorites");
        viewFaveButton.setBounds(485,0,150,25);
        viewFaveButton.addActionListener(e->{
            try {

```



```

        FavoritesWindow faveWindow = new FavoritesWindow(userid);
    } catch (IOException ioException) {
        ioException.printStackTrace();
    } catch (ClassNotFoundException classNotFoundException) {
        classNotFoundException.printStackTrace();
    }
    browseWindow.dispose();
});
browseWindow.add(viewFaveButton);

// Order Button
JButton orderButton = new JButton("View Orders");
orderButton.setBounds(0, 0, 150, 25);
orderButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            OrderWindow orderWindow = new OrderWindow(userID);
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } catch (ClassNotFoundException classNotFoundException) {
            classNotFoundException.printStackTrace();
        }

        browseWindow.dispose();
    }
});
browseWindow.add(orderButton);

DecimalFormat df = new DecimalFormat("#.00");
JLabel cartTotal = new JLabel("Total : $" + df.format(myCart.getTotalPrice()));
cartTotal.setBounds(640, 25, 150, 25);
browseWindow.add(cartTotal);

JButton checkoutButton = new JButton("Checkout");
checkoutButton.setBounds(640, 0, 150, 25);
checkoutButton.addActionListener((e)->{
    try {
        CheckoutWindow checkoutWindow=new CheckoutWindow(userID);
    } catch (Exception e1) {
        e1.printStackTrace();
    }
    browseWindow.dispose();
});

browseWindow.add(checkoutButton);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(10, 75, 760, 400);

```

```
browseWindow.add(scrollPane);
```

```
JPanel borderLayout = new JPanel();  
scrollPane.setViewportView(borderLayout);  
borderLayout.setLayout(new BorderLayout(0,0));
```

```
JPanel columnPanel = new JPanel();  
borderLayout.add(columnPanel, BorderLayout.NORTH);  
columnPanel.setLayout(new GridLayout(0, 1, 0, 1));  
columnPanel.setBackground(Color.gray);
```

```
SingletonProductList singleton = SingletonProductList.getInstance();  
Iterator<Product> listIter = singleton.getProductList().iterator();  
while(listIter.hasNext())  
{  
    productList.add(listIter.next());  
}
```

```
String[] sortStrings = {"Sort", "Name", "Price (High to Low)"};  
JComboBox sortList = new JComboBox(sortStrings);  
sortList.setSelectedIndex(0);  
sortList.setBounds(0, 40, 150, 25);  
browseWindow.add(sortList);  
sortList.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        SortByName sortByName = new SortByName();  
        JComboBox jcb = (JComboBox) e.getSource();  
        int strategySelected = jcb.getSelectedIndex();  
  
        if (strategySelected == 0) { //populate window with default (no sort)  
            for (int i = 0; i < productList.size(); i++)  
            {  
                JPanel rowPanel = new JPanel();  
                rowPanel.setPreferredSize(new Dimension(300, 200));  
                columnPanel.add(rowPanel);  
                rowPanel.setLayout(null);  
  
                JButton viewProductButton = new JButton(productList.get(i).getProductName());  
                viewProductButton.setBounds(50, 0, 150, 50);  
                final int j=i; //I feel dirty for doing this but I don't want to deal with it  
                viewProductButton.addActionListener(f -> {  
                    new ProductPopupWindow(productList.get(j));  
                });  
                rowPanel.add(viewProductButton);  
  
                // Add To Cart Button  
                JButton addToCartButton = new JButton("Add To Cart");  
                addToCartButton.setBounds(600, 0, 100, 50);
```

```

addToCartButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            myCart.addItem(productList.get(j));
            myCart.calculateTotalPrice();
            System.out.println("price: " + myCart.getTotalPrice());
            cartTotal.setText("Total: $" + df.format(myCart.getTotalPrice()));

            for(Product p: myCart.getCartContents())
            {
                System.out.println("in cart: " + p.getProductName());
            }
        } catch (IOException | ClassNotFoundException ioException) {
            ioException.printStackTrace();
        }
    }
});
rowPanel.add(addToCartButton);

```

```

JButton favoriteButton = new JButton("Add To Favorites");

```

```

System.out.println("\nFavorites:");
for (int k = 0; k < myFaveList.size(); k++)
{
    System.out.println(k + " : " + myFaveList.get(k).getProductName());

    if (productList.get(i).getProductID().equals(myFaveList.get(k).getProductID()))
    {
        favoriteButton.setText("Unfavorite");
        break;
    }
}

```

```

favoriteButton.setBounds(575, 100, 150, 50);
favoriteButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton clicked = (JButton) e.getSource();
        String clickedText = clicked.getText();
        System.out.println(clickedText);

        if (clickedText.equals("Add To Favorites"))
        {
            try {
                myFavorites.addFave(productList.get(j));
                clicked.setText("Unfavorite");
            } catch (IOException ioException) {

```

```

        ioException.printStackTrace();
    }
}
else
{
    try {
        for(Product p: myFavorites.getFaves())
        {
            myFavorites.removeFave(productList.get(j));
            clicked.setText("Add To Favorites");
        }
    } catch (IOException ioException) {
        ioException.printStackTrace();
    } catch (ClassNotFoundException classNotFoundException) {
        classNotFoundException.printStackTrace();
    }
}
}
});
rowPanel.add(favoriteButton);
//addListenerForAddToFav(addToFavButton, listIter.next());//productList.get(i));

// JLabel For Price
JLabel productPrice = new JLabel("Price : $" +
df.format(productList.get(i).getSellPrice()));//productList.get(i).getPrice());
productPrice.setBounds(50, 40, 100, 50);
rowPanel.add(productPrice);

// JLabel For Product Description
JLabel productDescription = new JLabel("<html>" + productList.get(i).getProductDescription() +
"</html>");//productList.get(i).getProductDescription()
productDescription.setBounds(50, 40, 500, 200);
rowPanel.add(productDescription);

JLabel quantityLabel = new JLabel("Quantity in Stock: " + productList.get(i).getAvailableQuantity());
quantityLabel.setBounds(50, 70, 200, 50);
rowPanel.add(quantityLabel);
}

browseWindow.setVisible(true);
}

else if (strategySelected == 1) { //populate contents sorted alphabetically by product name
    System.out.println("Strategy: " + strategySelected);
    Collections.sort(productList, sortByName.productSearch());
    columnPanel.removeAll();
    columnPanel.revalidate();
    //columnPanel.repaint();
    for (int i = 0; i < productList.size(); i++) {

```

```

JPanel rowPanel = new JPanel();
rowPanel.setPreferredSize(new Dimension(300, 200));
columnPanel.add(rowPanel);
rowPanel.setLayout(null);

JButton viewProductButton = new JButton(productList.get(i).getProductName());
viewProductButton.setBounds(50, 0, 150, 50);
final int j = i; //I feel dirty for doing this but I don't want to deal with it
viewProductButton.addActionListener(f -> {
    new ProductPopupWindow(productList.get(j));
});
rowPanel.add(viewProductButton);

// Add To Cart Button
JButton addToCartButton = new JButton("Add To Cart");
addToCartButton.setBounds(600, 0, 100, 50);
addToCartButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            myCart.addItem(productList.get(j));
            myCart.calculateTotalPrice();
            System.out.println("price: " + myCart.getTotalPrice());
            cartTotal.setText("Total: $" + df.format(myCart.getTotalPrice()));

            for (Product p : myCart.getCartContents()) {
                System.out.println("in cart: " + p.getProductName());
            }
        } catch (IOException | ClassNotFoundException ioException) {
            ioException.printStackTrace();
        }
    }
});
rowPanel.add(addToCartButton);

JButton favoriteButton = new JButton("Add To Favorites");

for (int k = 0; k < myFaveList.size(); k++) {
    if (productList.get(i).getProductID().equals(myFaveList.get(k).getProductID())) {
        favoriteButton.setText("Unfavorite");
        break;
    }
}

favoriteButton.setBounds(575, 100, 150, 50);
favoriteButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent c) {

```

```

        JButton clicked = (JButton) c.getSource();
        String clickedText = clicked.getText();
        System.out.println(clickedText);

        if (clickedText.equals("Add To Favorites")) {
            try {
                myFavorites.addFave(productList.get(j));
                clicked.setText("Unfavorite");
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
        } else {
            try {
                for (Product p : myFavorites.getFaves()) {
                    myFavorites.removeFave(productList.get(j));
                    clicked.setText("Add To Favorites");
                }
            } catch (IOException ioException) {
                ioException.printStackTrace();
            } catch (ClassNotFoundException classNotFoundException) {
                classNotFoundException.printStackTrace();
            }
        }
    }
});
rowPanel.add(favoriteButton);

JLabel productPrice = new JLabel("Price : $" +
df.format(productList.get(i).getSellPrice()); //productList.get(i).getPrice());
productPrice.setBounds(50, 40, 100, 50);
rowPanel.add(productPrice);

// JLabel For Product Description
JLabel productDescription = new JLabel("<html>" + productList.get(i).getProductDescription() +
"</html>"); //productList.get(i).getProductDescription()
productDescription.setBounds(50, 40, 500, 200);
rowPanel.add(productDescription);

JLabel quantityLabel = new JLabel("Quantity in Stock: " + productList.get(i).getAvailableQuantity());
quantityLabel.setBounds(50, 70, 200, 50);
rowPanel.add(quantityLabel);
}
browseWindow.setVisible(true);
}
else //populate contents sorted by price low to high
{
    SortByPrice sortByPrice = new SortByPrice();
    System.out.println("Strategy: " + strategySelected);
    Collections.sort(productList, sortByPrice.productSearch());
}

```

```

for(Product l : productList)
    System.out.println(l.getSellPrice());
columnPanel.removeAll();
columnPanel.revalidate();
for (int i = 0; i < productList.size(); i++) {

    JPanel rowPanel = new JPanel();
    rowPanel.setPreferredSize(new Dimension(300, 200));
    columnPanel.add(rowPanel);
    rowPanel.setLayout(null);

    JButton viewProductButton = new JButton(productList.get(i).getProductName());
    viewProductButton.setBounds(50, 0, 150, 50);
    final int j = i; //I feel dirty for doing this but I don't want to deal with it
    viewProductButton.addActionListener(f -> {
        new ProductPopupWindow(productList.get(j));
    });
    rowPanel.add(viewProductButton);

    // Add To Cart Button
    JButton addToCartButton = new JButton("Add To Cart");
    addToCartButton.setBounds(600, 0, 100, 50);
    addToCartButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                myCart.addItem(productList.get(j));
                myCart.calculateTotalPrice();
                System.out.println("price: " + myCart.getTotalPrice());
                cartTotal.setText("Total: $" + df.format(myCart.getTotalPrice()));

                for (Product p : myCart.getCartContents()) {
                    System.out.println("in cart: " + p.getProductName());
                }
            } catch (IOException | ClassNotFoundException ioException) {
                ioException.printStackTrace();
            }
        }
    });
    rowPanel.add(addToCartButton);

    JButton favoriteButton = new JButton("Add To Favorites");

    for (int k = 0; k < myFaveList.size(); k++) {
        if (productList.get(i).getProductID().equals(myFaveList.get(k).getProductID())) {
            favoriteButton.setText("Unfavorite");
            break;
        }
    }
}

```

```

favoriteButton.setBounds(575, 100, 150, 50);
favoriteButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent c) {
        JButton clicked = (JButton) c.getSource();
        String clickedText = clicked.getText();
        System.out.println(clickedText);

        if (clickedText.equals("Add To Favorites")) {
            try {
                myFavorites.addFave(productList.get(j));
                clicked.setText("Unfavorite");
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
        } else {
            try {
                for (Product p : myFavorites.getFaves()) {
                    myFavorites.removeFave(productList.get(j));
                    clicked.setText("Add To Favorites");
                }
            } catch (IOException ioException) {
                ioException.printStackTrace();
            } catch (ClassNotFoundException classNotFoundException) {
                classNotFoundException.printStackTrace();
            }
        }
    }
});
rowPanel.add(favoriteButton);

JLabel productPrice = new JLabel("Price : $" +
df.format(productList.get(i).getSellPrice())); //productList.get(i).getPrice());
productPrice.setBounds(50, 40, 100, 50);
rowPanel.add(productPrice);

// JLabel For Product Description
JLabel productDescription = new JLabel("<html>" + productList.get(i).getProductDescription() +
"</html>");//productList.get(i).getProductDescription()
productDescription.setBounds(50, 40, 500, 200);
rowPanel.add(productDescription);

JLabel quantityLabel = new JLabel("Quantity in Stock: " + productList.get(i).getAvailableQuantity());
quantityLabel.setBounds(50, 70, 200, 50);
rowPanel.add(quantityLabel);
}
browseWindow.setVisible(true);
}

```



```
}  
});
```

```
for (int i = 0; i < productList.size(); i++)  
{  
    JPanel rowPanel = new JPanel();  
    rowPanel.setPreferredSize(new Dimension(300, 200));  
    columnPanel.add(rowPanel);  
    rowPanel.setLayout(null);  
  
    JButton viewProductButton = new JButton(productList.get(i).getProductName());  
    viewProductButton.setBounds(50, 0, 150, 50);  
    final int j=i; //I feel dirty for doing this but I don't want to deal with it  
    viewProductButton.addActionListener(f -> {  
        new ProductPopupWindow(productList.get(j));  
    });  
    rowPanel.add(viewProductButton);  
  
    // Add To Cart Button  
    JButton addToCartButton = new JButton("Add To Cart");  
    addToCartButton.setBounds(600, 0, 100, 50);  
    addToCartButton.addActionListener(new ActionListener() {  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            try {  
                myCart.addItem(productList.get(j));  
                myCart.calculateTotalPrice();  
                System.out.println("price: " + myCart.getTotalPrice());  
                cartTotal.setText("Total: $" + df.format(myCart.getTotalPrice()));  
  
                for(Product p: myCart.getCartContents())  
                {  
                    System.out.println("in cart: " + p.getProductName());  
                }  
            } catch (IOException | ClassNotFoundException ioException) {  
                ioException.printStackTrace();  
            }  
        }  
    });  
    rowPanel.add(addToCartButton);  
  
    JButton favoriteButton = new JButton("Add To Favorites");  
  
    System.out.println("\nFavorites:");  
    for (int k = 0; k < myFaveList.size(); k++)  
    {  
        System.out.println(k + " : " + myFaveList.get(k).getProductName());  
    }  
}
```

```

        if (productList.get(i).getProductID().equals(myFaveList.get(k).getProductID()))
        {
            favoriteButton.setText("Unfavorite");
            break;
        }
    }
}

```

```

favoriteButton.setBounds(575, 100, 150, 50);
favoriteButton.addActionListener(new ActionListener() {

```

```

    @Override

```

```

    public void actionPerformed(ActionEvent e) {
        JButton clicked = (JButton) e.getSource();
        String clickedText = clicked.getText();
        System.out.println(clickedText);

```

```

        if (clickedText.equals("Add To Favorites"))
        {

```

```

            try {
                myFavorites.addFave(productList.get(j));
                clicked.setText("Unfavorite");
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
        }

```

```

        else
        {

```

```

            try {
                for(Product p: myFavorites.getFaves())
                {
                    myFavorites.removeFave(productList.get(j));
                    clicked.setText("Add To Favorites");
                }
            } catch (IOException ioException) {
                ioException.printStackTrace();
            } catch (ClassNotFoundException classNotFoundException) {
                classNotFoundException.printStackTrace();
            }
        }
    }
}

```

```

});

```

```

rowPanel.add(favoriteButton);

```

```

//addListenerForAddToFav(addToFavButton, listIter.next());//productList.get(i));

```

```

// JLabel For Price

```

```

JLabel productPrice = new JLabel("Price : $" +

```

```

df.format(productList.get(i).getSellPrice()));//productList.get(i).getPrice());

```

```

productPrice.setBounds(50, 40, 100, 50);

```

```

rowPanel.add(productPrice);

```

```

        // JLabel For Product Description
        JLabel productDescription = new JLabel("<html>" + productList.get(i).getProductDescription() +
"</html>");
        productDescription.setBounds(50, 40, 500, 200);
        rowPanel.add(productDescription);

        JLabel quantityLabel = new JLabel("Quantity in Stock: " + productList.get(i).getAvailableQuantity());
        quantityLabel.setBounds(50, 70, 200, 50);
        rowPanel.add(quantityLabel);
    }

    browseWindow.setVisible(true);
    browseWindow.setVisible(true);

}

private String userid;
private ArrayList<Product> productList = new ArrayList<>();
private ShoppingCart myCart;
private FavoritesList myFavorites;
private ArrayList<Product> myCartContents = new ArrayList<>();
private ArrayList<Product> myFaveList = new ArrayList<>();
}

```

## CheckoutWindow.java

```

package gui;

import java.text.DecimalFormat;
import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.*;
import java.io.IOException;
import java.util.ArrayList;
import client.ShoppingCart;
import client.ShoppingCart;
import client.Product;

/**
 * @author James Jenson, Jared Usher, Kirsten Hernquist
 * constructs and displays a checkout window.
 */
public class CheckoutWindow {
    public CheckoutWindow(String userID) throws IOException, ClassNotFoundException {
        this.userid = userID;
    }
}

```

```

this.myCart = new ShoppingCart(userid);
this.myCartContents = myCart.getCartContents();

for (Product p : myCartContents) {
    System.out.println("cart: " + p.getProductName());
}
myCart.calculateTotalPrice();
System.out.println("tot price: " + myCart.getTotalPrice());

final JFrame checkoutWindow = new JFrame("Checkout");
checkoutWindow.setBounds(100, 100, 800, 550);
checkoutWindow.setLayout(null);

JButton cancelButton = new JButton("Cancel");
cancelButton.setBounds(0, 0, 150, 25);
cancelButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            BrowseWindow browseWindow = new BrowseWindow(userID);
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } catch (ClassNotFoundException classNotFoundException) {
            classNotFoundException.printStackTrace();
        }

        checkoutWindow.dispose();
    }
});
checkoutWindow.add(cancelButton);

DecimalFormat df = new DecimalFormat("#.00");
JLabel cartTotal = new JLabel("Total : $" + df.format(myCart.getTotalPrice()));
cartTotal.setBounds(640, 25, 150, 25);
checkoutWindow.add(cartTotal);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(10, 75, 760, 400);
checkoutWindow.add(scrollPane);

JPanel borderLayout = new JPanel();
scrollPane.setViewPortView(borderLayout);
borderLayout.setLayout(new BorderLayout(0, 0));

JPanel columnPanel = new JPanel();
borderLayout.add(columnPanel, BorderLayout.NORTH);
columnPanel.setLayout(new GridLayout(0, 1, 0, 1));
columnPanel.setBackground(Color.gray);

```

```

int CartSize = myCartContents.size();

for (int i = 0; i < CartSize; i++) {
    JPanel rowPanel = new JPanel();
    rowPanel.setPreferredSize(new Dimension(300, 200));
    columnPanel.add(rowPanel);
    rowPanel.setLayout(null);

    JLabel productQuantity = new JLabel("In Cart: " + String.valueOf(myCart.getQuantity(myCartContents.get(i))));
    productQuantity.setBounds(50, 40, 200, 100);
    rowPanel.add(productQuantity);

    JButton addProductButton = new JButton("+");
    addProductButton.setBounds(700, 100, 25, 25);
    final int j = i;
    addProductButton.addActionListener(e -> {
        try {
            myCart.addItem(myCartContents.get(j));
            int currentQuantity = myCart.getQuantity(myCartContents.get(j));
            productQuantity.setText("In Cart: " + String.valueOf(currentQuantity++));
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    });
    rowPanel.add(addProductButton);

    JButton subtractProductButton = new JButton("-");
    subtractProductButton.setBounds(675, 100, 25, 25);
    final int k = i;
    subtractProductButton.addActionListener(e -> {
        try {
            //System.out.println("REMOVING NUMBER" + " " + myCart.getQuantity(myCartContents.get(k)));
            if (myCart.getQuantity(myCartContents.get(k)) > 1)
            {
                myCart.removeItem(myCartContents.get(k));
                //System.out.println("REMOVED PRODUCT");
                int currentQuantity = myCart.getQuantity(myCartContents.get(k));
                productQuantity.setText("In Cart: " + String.valueOf(currentQuantity--));
                //System.out.println("REMAINING NUMBER" + " " + myCart.getQuantity(myCartContents.get(k)) + "\n");
            }
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    });
    rowPanel.add(subtractProductButton);

    // JLabel For Price

```

```

        JLabel productPrice = new JLabel("Price : $" +
String.valueOf(myCartContents.get(i).getSellPrice())); //productList.get(i).getPrice());
        productPrice.setBounds(50, 40, 100, 50);
        rowPanel.add(productPrice);

        // JLabel For Product Description
        JLabel productName = new JLabel("<html>" + myCartContents.get(i).getProductName() +
"</html>");//productList.get(i).getDescription()
        productName.setBounds(50, 40, 500, 200);
        rowPanel.add(productName);

        if (myCart.getQuantity(myCartContents.get(i)) > 1)
        {
            i += myCart.getQuantity(myCartContents.get(i)) - 1;
        }
    }

    JButton confirmButton = new JButton("Confirm");
    confirmButton.setBounds(600, 0, 100, 50);
    confirmButton.addActionListener(e -> {
        try {
            PaymentWindow pay= new PaymentWindow(userID);
        } catch (Exception e1) {
            e1.printStackTrace();
        }

        checkoutWindow.dispose();

    });
    checkoutWindow.add(confirmButton);
    checkoutWindow.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    checkoutWindow.setVisible(true);
}

private String userID;
private ArrayList<Product> productList = new ArrayList<>();
private ShoppingCart myCart;
private ArrayList<Product> myCartContents = new ArrayList<>();
}

```

## **FavoritesWindow.java**

```

package gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import client.ShoppingCart;
import client.FavoritesList;
import client.FavoritesBundle;
import client.Product;
import client.DiscountedProduct;

/**
 * @author Jared User, James Jenson, Kirsten Hernquist
 * constructs a window for displaying a buyers favorite items, allows functionality to add to cart,
 * and allows functionality to add all items on the list to the cart.
 */
public class FavoritesWindow extends JFrame{

    public FavoritesWindow (String userID) throws IOException, ClassNotFoundException {

        this.userid = userID;
        FavoritesList myFavorites = new FavoritesList(userid);
        this.myFaves = myFavorites.getFaves();
        ShoppingCart myCart = new ShoppingCart(userid);

        final JFrame favoritesWindow = new JFrame("My Favorites");
        favoritesWindow.setBounds(100, 100, 800, 550);
        favoritesWindow.setLayout(null);

        // Order Button
        JButton backButton = new JButton("Back");
        backButton.setBounds(0, 0, 150, 25);
        favoritesWindow.add(backButton);

        backButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    BrowseWindow bw = new BrowseWindow(userid);
                    favoritesWindow.dispose();
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                } catch (ClassNotFoundException classNotFoundException) {
                    classNotFoundException.printStackTrace();
                }
            }
        });
    }
}

```

```

    }
});

JButton addAllButton = new JButton("Add All To Cart");
addAllButton.setBounds(175, 0, 150, 25);
addAllButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            FavoritesBundle myBundle = new FavoritesBundle(userID);
            for (Product p: myFaves)
            {
                myBundle.add(p);
            }
            Product d = new DiscountedProduct(myBundle.getProductName(), myBundle.getProductID(),
myBundle.getType(), myBundle.getSellPrice(),
                myBundle.getProductDescription(), myBundle.getAvailableQuantity(), myBundle.getInvoicePrice(),
myBundle.getSellerID());
            myCart.addItem(d);

            for(Product p : myCart.getCartContents())
            {
                System.out.println("in cart: " + p.getProductName());
            }

        } catch (ClassNotFoundException classNotFoundException) {
            classNotFoundException.printStackTrace();
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    }
});
favoritesWindow.add(addAllButton);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(10, 75, 760, 400);
favoritesWindow.add(scrollPane);

JPanel BorderLayout = new JPanel();
scrollPane.setViewPortView(BorderLayout);
BorderLayout.setLayout(new BorderLayout(0,0));

JPanel columnPanel = new JPanel();
BorderLayout.add(columnPanel, BorderLayout.NORTH);
columnPanel.setLayout(new GridLayout(0, 1, 0, 1));
columnPanel.setBackground(Color.gray);

for (Product p : myFaves)

```



```

{
    JPanel rowPanel = new JPanel();
    rowPanel.setPreferredSize(new Dimension(300, 200));
    columnPanel.add(rowPanel);
    rowPanel.setLayout(null);

    // Add To Cart Button
    JButton addToCartButton = new JButton("Add To Cart");
    addToCartButton.setBounds(600, 0, 100, 50);
    addToCartButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                myCart.addItem(p);
                myCart.calculateTotalPrice();

                for(Product p: myCart.getCartContents())
                {
                    System.out.println("in cart: " + p.getProductName());
                }
            } catch (IOException | ClassNotFoundException ioException) {
                ioException.printStackTrace();
            }
        }
    });
    rowPanel.add(addToCartButton);
    DecimalFormat df = new DecimalFormat("#.00");

    // JLabel For Price
    JLabel productPrice = new JLabel("Price : $" + df.format(p.getSellPrice()));
    productPrice.setBounds(50, 40, 100, 50);
    rowPanel.add(productPrice);

    JLabel nameLabel = new JLabel(p.getProductName());
    nameLabel.setBounds(50, 20, 100, 50);
    rowPanel.add(nameLabel);

    // JLabel For Product Description
    JLabel productDescription = new JLabel("<html>" + p.getProductDescription() + "</html>");
    productDescription.setBounds(50, 40, 500, 200);
    rowPanel.add(productDescription);
}

favoritesWindow.setVisible(true);
}

private ArrayList<Product> myFaves = new ArrayList();
private String userid;
}

```

## LoginWindow.java

```
package gui;
import client.UserDB;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;

/**
 * @author James Jenson
 * constructs and displays a login window.
 */
public class LoginWindow{

    public LoginWindow() {

        final JFrame loginWindow = new JFrame("Login");
        loginWindow.getContentPane().setLayout(new BorderLayout(loginWindow.getContentPane(), BorderLayout.Y_AXIS));
        JPanel panel = new JPanel();

        JLabel usernameLabel = new JLabel("Username: ");
        JTextField usernameField = new JTextField(0);
        usernameField.setPreferredSize(new Dimension(400,20));
        usernameField.setMaximumSize( usernameField.getPreferredSize() );

        JLabel passwordLabel = new JLabel("Password: ");
        JTextField passwordField = new JTextField(0);
        passwordField.setPreferredSize(new Dimension(400,20));
        passwordField.setMaximumSize(passwordField.getPreferredSize() );

        JPanel usernamePanel= new JPanel();
        usernamePanel.setLayout(new BorderLayout(usernamePanel, BorderLayout.X_AXIS));
        usernamePanel.add(usernameLabel);
        usernamePanel.add(usernameField);

        JPanel passwordPanel= new JPanel();
        passwordPanel.setLayout(new BorderLayout(passwordPanel, BorderLayout.X_AXIS));
        passwordPanel.add(passwordLabel);
        passwordPanel.add(passwordField);

        loginWindow.add(usernamePanel);
        JPanel buffer= new JPanel();
        buffer.setPreferredSize(new Dimension(4,2));
        loginWindow.add(buffer);
```

```
loginWindow.add(passwordPanel);
loginWindow.getContentPane().add(panel);
```

```
JButton submit = new JButton("Submit");
submit.addActionListener(new ActionListener() {
```

```
    @Override
```

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

```
        UserDB verify = new UserDB();
```

```
        String b = verify.verifyLogin(usernameField.getText(), passwordField.getText());
```

```
        System.out.println("b" + b);
```

```
        if (b!=null)
```

```
        {
```

```
            if (b.endsWith("s"))
```

```
            {System.out.println("successful login");
```

```
                try {
```

```
                    SellerInventoryWindow sPage = new SellerInventoryWindow(b);
```

```
                    loginWindow.dispose();
```

```
                } catch (IOException ioException) {
```

```
                    ioException.printStackTrace();
```

```
                } catch (ClassNotFoundException classNotFoundException) {
```

```
                    classNotFoundException.printStackTrace();
```

```
                }
```

```
            }
```

```
            else if (b.endsWith("b"))
```

```
            {
```

```
                System.out.println("ends with b");
```

```
                try {
```

```
                    BrowseWindow buyerWindow = new BrowseWindow(b);
```

```
                    loginWindow.dispose();
```

```
                } catch (IOException ioException) {
```

```
                    ioException.printStackTrace();
```

```
                } catch (ClassNotFoundException classNotFoundException) {
```

```
                    classNotFoundException.printStackTrace();
```

```
                }
```

```
            }}
```

```
        else{
```

```
            JOptionPane.showMessageDialog(new JFrame(), "Invalid Login");
```

```
        }
```

```
    }
```

```
});
```

```
loginWindow.add(submit);
```

```
loginWindow.setSize(455, 150);
```

```
loginWindow.setResizable(false);
```

```
loginWindow.setVisible(true);
```

```
}
```

```
    private String userID;  
}
```

## NewItemWindow.java

```
package gui;  
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.IOException;  
import java.util.ArrayList;  
import client.Product;  
import client.SellerFinancialView;  
import client.SellerInventory;  
import client.SingletonProductList;  
  
/**  
 * constructs and displays a window for a seller to add a new item.  
 * @author Kirsten Hernquist  
 */  
public class NewItemWindow extends JFrame{  
  
    public NewItemWindow(String sellerID, SellerFinancialView sfv) throws IOException, ClassNotFoundException {  
        this.sellerid = sellerID;  
        this.sView = sfv;  
  
        final JFrame frame = new JFrame("New Listing");  
        frame.setBounds(100,100,800,575);  
  
        JLabel titleLabel = new JLabel("New Listing");  
        frame.add(titleLabel, BorderLayout.NORTH);  
  
        JLabel nameLabel = new JLabel("Product Name: ");  
        JTextField nameField = new JTextField();  
  
        JLabel idLabel = new JLabel("Product ID: ");  
        JTextField idField = new JTextField();  
  
        JLabel typeLabel = new JLabel("Product Type: ");  
        JTextField typeField = new JTextField();  
  
        JLabel salePriceLabel = new JLabel("Sale Price: ");  
        JTextField salePriceField = new JTextField();  
  
        JLabel descLabel = new JLabel("Description: " );
```

```

TextField descField = new TextField();

JLabel availableLabel = new JLabel("Quantity Available: ");
TextField availableField = new TextField();

JLabel invPriceLabel = new JLabel("Invoice Price: ");
TextField invPriceField = new TextField();

JPanel formPanel = new JPanel();
formPanel.setLayout(new GridLayout(7, 2, 1, 1));

formPanel.add(nameLabel);
formPanel.add(nameField);
formPanel.add(idLabel);
formPanel.add(idField);
formPanel.add(typeLabel);
formPanel.add(typeField);
formPanel.add(salePriceLabel);
formPanel.add(salePriceField);
formPanel.add(descLabel);
formPanel.add(descField);
formPanel.add(availableLabel);
formPanel.add(availableField);
formPanel.add(invPriceLabel);
formPanel.add(invPriceField);

frame.add(formPanel, BorderLayout.CENTER);

JButton submitButton = new JButton("Submit");

SellerInventory inventory = new SellerInventory(sellerid);

submitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            SingletonProductList spl = SingletonProductList.getInstance();
            double salePrice = Double.parseDouble(salePriceField.getText());
            double invPrice = Double.parseDouble(invPriceField.getText());
            int available = Integer.parseInt(availableField.getText());
            Product newProduct = new Product(nameField.getText(), idField.getText(), typeField.getText(), salePrice,
descField.getText(),
                available, invPrice, sellerid);

            spl.add(newProduct);
            spl.saveList(spl.getList(sellerid));

            inventory.addChangeListener(sView);
            inventory.calculateCosts();

```

```

        SellerInventoryWindow window = new SellerInventoryWindow(sellerid);

        frame.dispose();

        } catch (IOException ioException) {
            ioException.printStackTrace();
        } catch (ClassNotFoundException classNotFoundException) {
            classNotFoundException.printStackTrace();
        }
    }
});

frame.add(submitButton, BorderLayout.SOUTH);
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
frame.setVisible(true);
}

private String sellerid;
final SellerFinancialView sView;
}

```

## OrderWindow.java

```

package gui;
import client.Order;
import client.OrderList;
import client.Product;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.io.IOException;
import java.text.DecimalFormat;

import java.util.ArrayList;

/**
 * @author Jared Usher
 */
public class OrderWindow
{
    /**
     * Creates And Displays A Window Containing All Of A User's
     * Previous Orders
     * @param userID ID of Buyer Account Currently Signed In
     */
}

```

```

* @throws IOException
* @throws ClassNotFoundException
*/
public OrderWindow(String userID) throws IOException, ClassNotFoundException {
    this.userid = userID;
    this.myOrders = new OrderList(userID);
    this.myOrderList = myOrders.getOrders();

    DecimalFormat df = new DecimalFormat("#.00");

    final JFrame orderWindow = new JFrame("Past Orders");
    orderWindow.setBounds(100, 100, 800, 550);
    orderWindow.setLayout(null);

    /**
     * Creates And Adds A Back Button With An ActionListener
     * To Return Back To The BrowseWindow
     */
    JButton backButton = new JButton("Back");
    backButton.setBounds(0, 0, 150, 25);
    orderWindow.add(backButton);

    backButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                BrowseWindow buyerWindow = new BrowseWindow(userID);
            } catch (IOException ioException) {
                ioException.printStackTrace();
            } catch (ClassNotFoundException classNotFoundException) {
                classNotFoundException.printStackTrace();
            }

            orderWindow.dispose();
        }
    });

    /**
     * Creates And Adds A Scroll-able UI To Contain The Orders
     * In Order To Account For Any Number Of Orders To Be Displayed
     */
    JScrollPane scrollPane = new JScrollPane();
    scrollPane.setBounds(10, 75, 760, 400);
    orderWindow.add(scrollPane);

    JPanel BorderLayout = new JPanel();
    scrollPane.setViewPortView(BorderLayout);
    BorderLayout.setLayout(new BorderLayout(0,0));

```

```

JPanel columnPanel = new JPanel();
borderLayout.add(columnPanel, BorderLayout.NORTH);
columnPanel.setLayout(new GridLayout(0, 1, 0, 1));
columnPanel.setBackground(Color.gray);

/**
 * Iterates Through The User's Previous Orders
 * Creating A JPanel TO Display The Information For Each One
 */
for (int i = 0; i < myOrderList.size(); i++)
{
    ArrayList<Product> orderProducts = myOrderList.get(i).getOrderProducts();

    JPanel rowPanel = new JPanel();
    if (orderProducts.size() == 1)
    {
        rowPanel.setPreferredSize(new Dimension(300, 60 * 3));
    }
    else
    {
        rowPanel.setPreferredSize(new Dimension(300, 60 * (orderProducts.size())));
    }
    columnPanel.add(rowPanel);
    rowPanel.setLayout(null);

    JLabel itemText = new JLabel("Order Items : ");
    Font font = itemText.getFont();
    Map attributes = font.getAttributes();
    attributes.put(TextAttribute.UNDERLINE, TextAttribute.UNDERLINE_ON);
    itemText.setFont(font.deriveFont(attributes));
    itemText.setBounds(50, 20, 200, 50);
    rowPanel.add(itemText);

    JLabel productText;

    /**
     * Iterates Through Each Product Within An Order
     * Adding JLabels Containing The Product Name And Price
     * To The JPanel
     */
    for (int j = 0; j < orderProducts.size(); j++)
    {
        productText = new JLabel(orderProducts.get(j).getProductName() +
            " | $" + df.format(orderProducts.get(j).getSellPrice()));
        productText.setBounds(50, 50 + (40 * j), 250, 50);
        rowPanel.add(productText);
    }
}

```



```

        // JLabel For Total Price Of An Order
        JLabel orderPrice = new JLabel("Order Total : $" +
            df.format(myOrderList.get(i).getOrderPrice()));
        orderPrice.setBounds(400, 50, 200, 50);
        rowPanel.add(orderPrice);
    }

    orderWindow.setVisible(true);
}

private String userid;
private OrderList myOrders;
private ArrayList<Order> myOrderList = new ArrayList<>();
}

```

## PaymentWindow.java

```

package gui;

import javax.swing.*;
import client.Order;
import client.OrderList;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import client.ShoppingCart;
import client.Product;
import client.SingletonProductList;
import java.util.ArrayList;

/**
 * @author Kirsten Hernquist, James Jenson, Jared Usher
 * Constructs and displays a payment window for the user to enter payment information.
 */
public class PaymentWindow {

    public PaymentWindow(String userid) throws IOException, ClassNotFoundException {

        this.userID = userid;
        this.myCart = new ShoppingCart(userID);
        this.myCartContents = myCart.getCartContents();
        SingletonProductList products = SingletonProductList.getInstance();
        productList = products.getProductList();

        for(Product c: myCart.getCartContents())

```

```

{
    System.out.println("in cart " + c.getProductName() + " " + myCart.getQuantity(c));
}

final JFrame paymentWindow = new JFrame("Checkout");
paymentWindow.getContentPane().setLayout(new BorderLayout(paymentWindow.getContentPane(),
BoxLayout.Y_AXIS));

JPanel panel = new JPanel();

JLabel ccNumberLabel = new JLabel("Credit Card Number: ");
JTextField ccNumberField = new JTextField(0);
ccNumberField.setPreferredSize(new Dimension(400,20));
ccNumberField.setMaximumSize(ccNumberField.getPreferredSize() );

JLabel secNumLabel = new JLabel("CVV: ");
JTextField secNumField = new JTextField(0);
secNumField.setPreferredSize(new Dimension(400,20));
secNumField.setMaximumSize(secNumField.getPreferredSize() );

JPanel ccNumPanel= new JPanel();
ccNumPanel.setLayout(new BorderLayout(ccNumPanel, BorderLayout.X_AXIS));
ccNumPanel.add(ccNumberLabel);
ccNumPanel.add(ccNumberField);

JPanel secNumPanel= new JPanel();
secNumPanel.setLayout(new BorderLayout(secNumPanel, BorderLayout.X_AXIS));
secNumPanel.add(secNumLabel);
secNumPanel.add(secNumField);

paymentWindow.add(ccNumPanel);
JPanel buffer= new JPanel();
buffer.setPreferredSize(new Dimension(4,2));
paymentWindow.add(buffer);
paymentWindow.add(secNumPanel);
paymentWindow.getContentPane().add(panel);

JButton submit = new JButton("Confirm");
submit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {

            System.out.println("Mycartcontents size: "+ myCart.getCartContents().size());

            for (int k = 0; k < myCartContents.size(); k++)
            {

                SingletonProductList si = SingletonProductList.getInstance();

```

```
si.sellQuantity(myCartContents.get(k).getProductID(), 1);
```

```
try {
    Order newOrder = new Order(myCart);
    OrderList myOrders = new OrderList(userID);
    myOrders.addOrder(newOrder);
    SingletonProductList singleton = SingletonProductList.getInstance();

    for (Product p : myCart.getCartContents())
    {
        singleton.updateAvailableQuantity(p.getProductID(), p.getAvailableQuantity()-
myCart.getQuantity(p));
    }
} catch (IOException ioException) {
    ioException.printStackTrace();
} catch (ClassNotFoundException classNotFoundException) {
    classNotFoundException.printStackTrace();
} catch (Exception e2) {
    e2.printStackTrace();
}
}
} catch (IOException ioException) {
    ioException.printStackTrace();
} catch (ClassNotFoundException classNotFoundException) {
    classNotFoundException.printStackTrace();
}

try {
    myCart.emptyCart();
    BrowseWindow buyerWindow = new BrowseWindow(userID);
    paymentWindow.dispose();
}
catch (Exception e3) { e3.printStackTrace();}
}};
```

```
JButton cancel = new JButton("Cancel");
```

```
cancel.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            BrowseWindow bw = new BrowseWindow(userID);
            paymentWindow.setVisible(false);
            paymentWindow.dispose();
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } catch (ClassNotFoundException classNotFoundException) {
```

```

        classNotFoundException.printStackTrace();
    }
    });

    paymentWindow.add(cancel);
    paymentWindow.add(submit);
    paymentWindow.setSize(455, 150);
    paymentWindow.setResizable(false);
    paymentWindow.setVisible(true);
    paymentWindow.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
private ArrayList<Product> myCartContents= new ArrayList<>();
private String userID;
private ShoppingCart myCart;
private ArrayList<Product> productList = new ArrayList<>();
}

```

## ProductPopUpWindow.java

```

package gui;

import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import client.SingletonProductList;
import client.Product;

/**
 * @author James Jenson
 * constructs and displays a product pop up window with more in depth information about the product.
 */
public class ProductPopupWindow {
    public ProductPopupWindow(Product p) {
        final JFrame window = new JFrame("Popup");

        JPanel panel1= new JPanel();
        panel1.setLayout(new BoxLayout(panel1, BoxLayout.Y_AXIS));

        JLabel name= new JLabel(p.getProductName());
        panel1.add(name);

        JLabel desc= new JLabel("Product description: "+p.getProductDescription());
        panel1.add(desc);

        panel1.add(new JLabel("Quantity in stock: "+p.getAvailableQuantity() ));
    }
}

```

```

        window.add(panel1);
        window.setSize(455, 150);
        window.setResizable(false);
        window.setVisible(true);
    }
}

```

## **SellerFinancialView.java**

```
package client;
```

```

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.*;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Dimension;

```

```

/**
 * @author Kirsten Hernquist
 * a class that constructs and updates the seller financial information.
 */

```

```
public class SellerFinancialView extends JPanel implements ChangeListener {
```

```

    /**
     * constructs the seller financial information.
     * @param cost the costs incurred by the seller.
     * @param revenue the revenue made by the seller.
     * @param profit the profits made by the seller.
     * @param color the color to change the text upon changes to cost.
     */

```

```

    public SellerFinancialView(double cost, double revenue, double profit, Color color) {
        this.cost = cost;
        this.revenue = revenue;
        this.profit = profit;
    }

```

```

        JPanel financePanel = new JPanel();
        financePanel.setPreferredSize(new Dimension(375, 100));
    }

```

```

        JLabel titleLabel = new JLabel("Finances");
        financePanel.add(titleLabel);
    }

```

```

        JLabel costLabel = new JLabel("Costs: " + cost);
        this.costLabel = costLabel;
        financePanel.add(costLabel);
    }

```

```

JLabel revLabel = new JLabel("Revenue: " + revenue);
this.revLabel = revLabel;
financePanel.add(revLabel);

JLabel profLabel = new JLabel("Profits: " + profit);
this.profitLabel = profLabel;
financePanel.add(profLabel);

add(financePanel);
}

/**
 * paints and repaints the cost label
 * @param g, the Graphics
 */
@Override
public void paint(Graphics g) {
    super.paint(g);
    costLabel.setText("Costs: " + String.valueOf(cost));
    revLabel.setText("Revenue: " + String.valueOf(revenue));
    profitLabel.setText("Profits: " + String.valueOf(profit));
    costLabel.setForeground(this.color);
    g.setColor(this.color);
}

/**
 * tells the financial view that the state has changed.
 * @param c the changeevent to process.
 * @precondition: a change in the inventory has occurred.
 * @postcondition: the cost label color and text changes.
 */
@Override
public void stateChanged(ChangeEvent c) {
    SellerInventory source = (SellerInventory) c.getSource();
    if (cost < source.getCosts()) {
        this.color = java.awt.Color.RED;
    }
    else {
        this.color = java.awt.Color.GREEN;
    }
    this.cost = source.getCosts();
    this.revenue = source.getRevenue();
    this.profit = source.getProfit();
    this.repaint();
}

private Color color;
private double cost;

```

```

private JLabel costLabel;
private double revenue;
private double profit;
private JLabel profitLabel;
private JLabel revLabel;
}

```

## SellerInventory.java

```

package gui;

```

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.ArrayList;
import client.Product;
import client.SingletonProductList;
import client.SellerInventory;
import client.SellerFinancialView;

```

```

/**

```

```

 * constructs and displays a seller inventory window, where a seller can view their inventory

```

```

 * and financial details.

```

```

 * @author Kirsten Hernquist

```

```

 */

```

```

public class SellerInventoryWindow extends JFrame{

```

```

    public SellerInventoryWindow(String SellerID) throws IOException, ClassNotFoundException {
        this.sellerid = SellerID;

```

```

        SingletonProductList globalSingleton = SingletonProductList.getInstance();

```

```

        ArrayList<Product> globalList = new ArrayList<>();

```

```

        globalList = globalSingleton.getList(sellerid);

```

```

        SellerInventory myInventory = new SellerInventory(sellerid);

```

```

        myProducts = myInventory.getInventory();

```

```

        this.sellerid = SellerID;

```

```

        final JFrame frame = new JFrame("Inventory");

```

```

        frame.setBounds(100, 100, 800, 550);

```

```

        frame.setDefaultCloseOperation(EXIT_ON_CLOSE);

```

```

        frame.setLayout(new FlowLayout());

```

```

JLabel sellerIdLabel= new JLabel("Seller ID: " + sellerid);
frame.add(sellerIdLabel);//, BorderLayout.WEST);

JLabel inventoryLabel = new JLabel("Current Inventory");
frame.add(inventoryLabel);

JPanel productPanel = new JPanel(); //where other panels will go
productPanel.setSize(475, 300);
productPanel.setLayout(new BoxLayout(productPanel, BoxLayout.Y_AXIS));

myInventory.calculateProfits();
myInventory.calculateRevenue();
myInventory.calculateCosts();
SellerFinancialView newFV = new SellerFinancialView(myInventory.getCosts(), myInventory.getRevenue(),
myInventory.getProfit(), Color.BLACK);
JButton listingButton = new JButton("Add New Listing");
listingButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            NewItemWindow itemWindow = new NewItemWindow(sellerid, newFV);
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } catch (ClassNotFoundException classNotFoundException) {
            classNotFoundException.printStackTrace();
        }
        frame.setVisible(false);
        frame.dispose();
    }
});
productPanel.add(listingButton);

for(int i = 0; i < myProducts.size(); i++)
{
    int k = i;
    JPanel newProduct = new JPanel();
    newProduct.setPreferredSize(new Dimension(300, 200));
    newProduct.setLayout(new GridLayout(4, 2, 1, 1));
    JLabel nameLabel = new JLabel(myProducts.get(i).getProductName());
    newProduct.add(nameLabel);

    JLabel idLabel = new JLabel("Product ID: " + myProducts.get(i).getProductID());
    newProduct.add(idLabel);
    newProduct.add(idLabel);
    JLabel typeLabel = new JLabel("Type: " + myProducts.get(i).getType());
    newProduct.add(typeLabel);
    JLabel quantityLabel = new JLabel("Quantity: " + myProducts.get(i).getAvailableQuantity());
    newProduct.add(quantityLabel);
    JLabel invoicePricelabel = new JLabel("Invoice Price: " + myProducts.get(i).getInvoicePrice());

```



```

newProduct.add(invoicePricelabel);
JLabel sellingPriceLabel = new JLabel("Selling Price: " + myProducts.get(i).getSellPrice());
newProduct.add(sellingPriceLabel);
JButton increaseQuantityButton = new JButton("+");

```

```

increaseQuantityButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        try {
            globalSingleton.addOrSubtractQuantity(myProducts.get(k).getProductID(), 1);
            ArrayList<Product> updatedStuff = globalSingleton.getList(sellerid);
            SellerInventory newInventory = new SellerInventory(sellerid);
            newInventory.addChangeListener(new FV);
            String newquantity = String.valueOf(updatedStuff.get(k).getAvailableQuantity());
            quantityLabel.setText("Quantity: " + newquantity);
            newInventory.calculateCosts();
            newInventory.calculateRevenue();
            newInventory.calculateProfits();

        } catch (IOException | ClassNotFoundException ioException) {
            ioException.printStackTrace();
        }

    }
});
newProduct.add(increaseQuantityButton);

```

```

JButton decreaseQuantityButton = new JButton("-");
decreaseQuantityButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            globalSingleton.addOrSubtractQuantity(myProducts.get(k).getProductID(), -1);
            ArrayList<Product> updatedStuff = globalSingleton.getList(sellerid);
            SellerInventory newInventory = new SellerInventory(sellerid);

            newInventory.addChangeListener(new FV);

            newInventory.calculateCosts();
            newInventory.calculateRevenue();
            newInventory.calculateProfits();

            String newquantity = String.valueOf(updatedStuff.get(k).getAvailableQuantity());
            quantityLabel.setText("Quantity: " + newquantity);

        } catch (IOException | ClassNotFoundException ioException) {
            ioException.printStackTrace();
        }
    }
});

```

```

    }
    }
});
newProduct.add(decreaseQuantityButton);

productPanel.add(newProduct);
}

JScrollPane productPane = new JScrollPane(productPanel);//productPanel);
productPane.setPreferredSize(new Dimension(700,400));
productPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
productPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

frame.getContentPane().add(productPane);
frame.getContentPane().add(newFV);

JButton updateButton = new JButton("update");
updateButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        productPanel.repaint();

    }
});

frame.setVisible(true);
}

private ArrayList<JPanel> products = new ArrayList<>();
private ArrayList<Product> myProducts = new ArrayList<>();
private String sellerid;
private ArrayList<Product> newInventory = new ArrayList<>();
private final ArrayList<Product> updatedInventory = new ArrayList<>();
}

```

## Glossary

- **Transaction:** An instance of buying or selling product(s).
- **Product:** An item for sale containing a product description, pricing, and availability.
- **Inventory:** Goods a seller lists to sell.
- **Seller:** A third-party user whose intent is to sell products.
- **Buyer:** A user whose intent is to purchase products.

- **Order:** A bundle of items that has been paid for by the buyer.
- **Cart:** A list of products the buyer wishes to purchase.
- **System:** A shopping cart application that executes the exchange of money for products between a buyer and a seller.
- **Item Counts/Quantity:** The number of a certain product that is available to be sold and purchased.
- **Seller Homepage:** The homepage that displays features specific to a seller.
- **Buyer Homepage:** The homepage that displays features specific to a buyer.
- **Browse:** Scroll through products.

### **Actors**

- Buyers
- Sellers