COP 4331-002
S Hassan Shaikh, Austin Vesquez, Jorge Martinez, Divyesh Mangapuram
12/07/2023
Option A: Shopping Cart Design Specifications

Shopping Cart Application

Functional specifications

The app allows a user to register or log in either as a customer or a seller. Sellers can sell new products and review or update existing products. Customers can review products, add them to their cart, review or update their shopping carts, and check out.

The platform the Shopping Cart Application will be developed on is Swing UI (User Interface).

Glossary

Account: An ideally unique identifier for an individual user, containing their information.

Class: A representation of an object in programming.

CRC: Class, responsibilities, and collaborators; defines the responsibilities of a class and what classes it works with to achieve those responsibilities.

Credentials: Username, password, and email address; The proof an individual owns an account.

Customer: An individual who serves the purpose of purchasing products.

Inventory: The stock of products a seller has for offer.

Item: An unspecified material object.

Login: The act of entering an application with an individual account.

Password: A private key word used to gain access to an account associated with a username.

Product: An item put up for sale.

Registration: The act of creating an account for an application.

Seller: An individual who serves the purpose of offering and selling products.

Sequence: A series of events resulting in an action.

Shopping Cart: A container for all the items that a customer is planning to purchase.

Stakeholder: A person or group with involvement in or control over the design decisions of the software such as developers and sponsors.

State: The current environment and situation.

Specification: The required functionality posed by the stakeholders.

UI: User Interface; the front for the software that the user interacts with.

UML: Unified Modelling Language; a method of representing classes, sequences, states, etc. Via graphs when designing software.

Use Case: A scenario about a user’s interaction with the software and the step-by-step protocols to consider.

User: The individual who interacts with the software.

Username: A public identifier for a user’s account.

UX: User experience; the experience of the user’s interaction with the software, this includes UI.

Use Cases

Use Case: User Logs In

Actor:

User

Preconditions:

The user has already registered an account on the shopping cart app.

The app is accessible and running.

Main Flow:

1. The user opens the shopping cart app.
2. The app displays the login screen, which includes fields for the user to enter their login credentials:
   * Username
   * Password
3. The user enters their registered username and password.
4. The app validates the entered credentials:
   * The app checks if the username exists in the database.
   * The app verifies that the entered password matches the stored password for the corresponding user account.
5. User logs in
   * The app logs the user in.
   * The user is redirected to the app's home screen after a successful login.

Alternative Flow:

5a. User inputs incorrect credentials
   * The app displays an error message ("Incorrect credentials.").
   * The user is prompted to re-enter their login credentials.

Use Case: User Changes/Updates their Login Credentials and Balance

Actor:

User

Preconditions:

The user has already logged in to their account.

The app is accessible and running.

Main Flow:

1. The user navigates to the "User Settings" page.
2. The user can update their name, username, password, and add to their balance.
3. The app validates the information.
4. If the information is valid, the app updates the user information and returns to user settings screen.

Alternative Flow:

4a. If the user did not enter a valid fund amount, all other changes are saved and displays an error message pertaining to an invalid fund field.

Use Case: Customer Adds Items to Shopping Cart

Actor:

User

Preconditions:

Items have been added to the database and marked for sale by a seller.

User has shopping cart app open and is logged in.

User has the shopping menu open.

User has added enough balance to afford the items for purchase

Main Flow:

1. User selects an available item.
2. The user can choose how many of the item they want.
3. Item is added to the user's shopping cart.
4. User enters into the shopping cart to checkout.
5. User chooses to complete transaction, a receipt is displayed and sends user to the shopping cart menu.

Alternative Flow:

4a. User can edit quantity amounts

5a. User chooses to continue shopping without completing transaction and returns to step 1.

Exceptions:

If the user enters a quantity greater than the stock number display, “The amount is over the stock of the item” and return to item number selection.

If the user enters an invalid character or a non-integer value, display “Invalid stock input” and return to item number selection.

If the user enters a number below 0, display “When adding a new item stock must be greater than 0”

Use Case: Customer Reviews Product Details

Actor:

Customer

Preconditions:

Items have been added to the database and marked for sale by a seller.

User has shopping cart app open and is logged in.

User has the shopping menu open.

Main Flow:

1. User selects an available item.
2. User selects to view details.
3. App displays a view showing important details about item.

Use Case: Customer Reviews/Updates Shopping Cart

Primary Actor:

Customer

Stakeholders:

Customer: Wants to review and modify the shopping cart.

Seller: Wants to ensure inventory is updated based on customer actions.

Preconditions:

The customer is logged in and has items in the shopping cart.

Main Flow:

1. The customer clicks on the shopping cart button.
2. The system displays the items in the cart, each with a product name, seller name, price, quantity, and subtotal.
3. The customer adjusts the quantity of an item using input boxes.
4. The customer clicks the complete transaction button.
5. The system recalculates the subtotal for each item and the total amount and prints the receipt.

6. Optionally, the customer removes items by clicking a remove button, and the system updates the total amount.

Postconditions:

The shopping cart reflects the modifications, and the total amount is recalculated.

Exceptions:

If the customer tries to increase/decrease the quantity beyond availability or enters an invalid input, the system shows an error message.

Use Case: Customer Checks Out

Primary Actor:

Customer

Stakeholders:

Customer: Wants a smooth and secure checkout process.

Seller: Wants to ensure inventory is accurately updated and payment is securely processed.

Preconditions:

The customer is logged in, has enough balance added, has finalized their shopping cart, and is ready to make a payment.

Main Flow:

1. The customer clicks on the proceed to checkout button.
2. The system displays a summary of items and total amount.
3. The customer clicks the pay button.
4. The system validates there is enough balance in the account.
5. The system updates the inventory, reducing the quantity of purchased items and subtracts subtotal from customer balance.

Postconditions:

The inventory is updated, the transaction is recorded, and a confirmation is sent to the customer.

Exceptions:

If the payment fails, the system shows an error message and asks the customer to enter the details again.

Use Case: Seller Reviews/Updates Inventory

Actor:

Seller

Preconditions:

The seller has already logged in to their seller account.

The app is accessible and running.

Main Flow:

1. The app displays the current inventory status, product name, price, product availability, and stock levels.
2. The seller can update product details by clicking on an item and edit name, description, price, stock, ability to permit sales, and custom fields.
3. The app updates the inventory with the changes made by the seller.
4. The app displays a success message ("Action was successful").

Alternative Flow:

3a. If there are issues with updating the inventory, such as invalid values or prompts, the app displays an error message and restarts the flow from 2.

Use Case: Seller Adds New Product

Actor:

Seller

Preconditions:

The seller has already logged in to their seller account.

The app is accessible and running.

Main Flow:

5. The seller navigates to the "Add New Item" page.
6. The seller provides product details, including name, description, price, quantity, stock, and if the item is on sale.
7. The app validates the information.
8. If the information is valid, the app adds the new product to the catalog.
9. The app displays a success screen, showing the item details and the added item in the catalog.

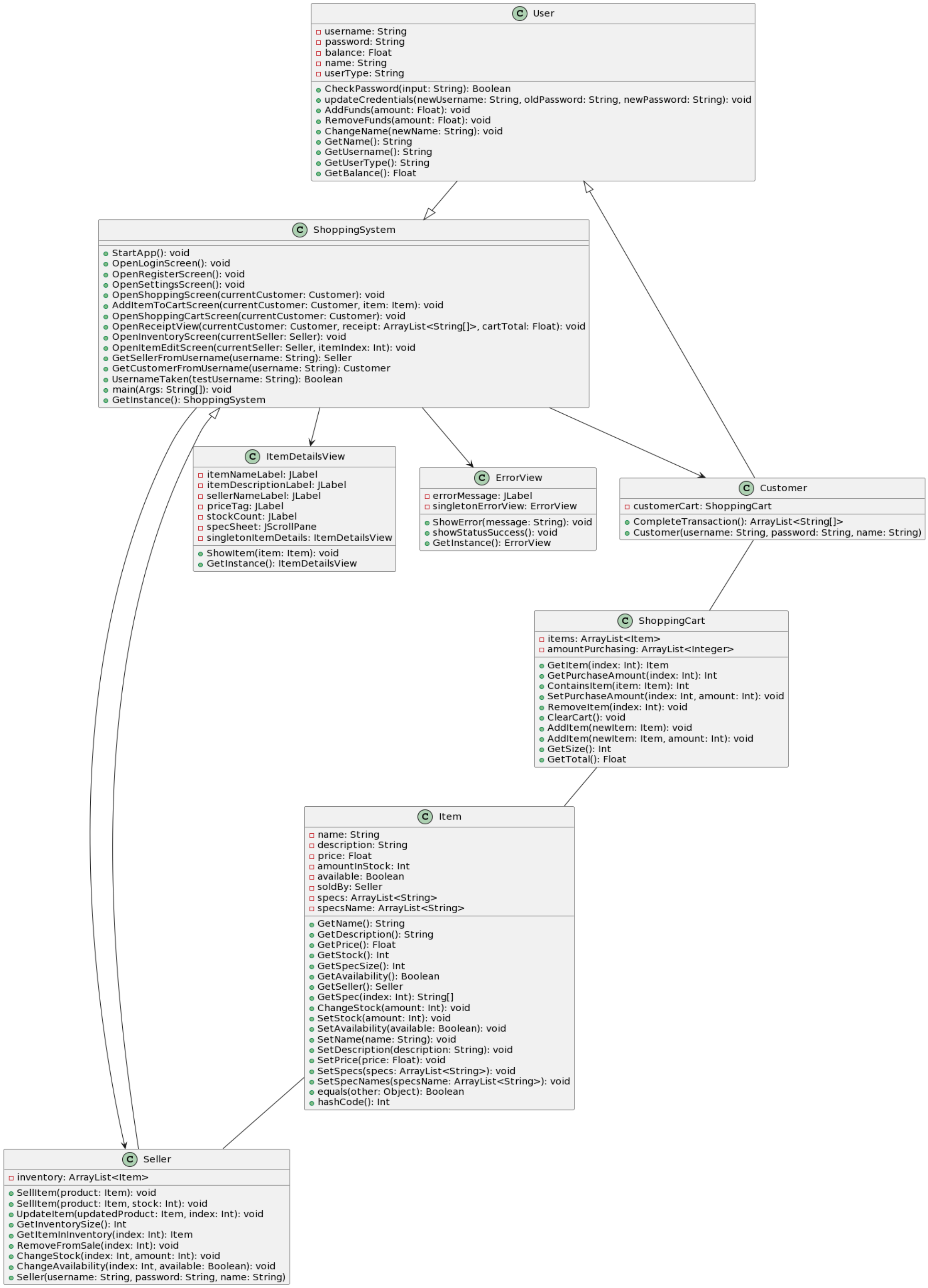
Alternative Flow:

4a. If there are issues with adding the new product, such as validation errors, the app displays an error message and prompts the seller to correct the information.

CRC Cards

<table>
<tr><th colspan="2">User</th></tr>
<tr><th>Responsibilities</th><th>Collaborators</th></tr>
<tr><td>Store and update login credentials</td><td>Customer</td></tr>
<tr><td>Store and update basic profile information</td><td>Seller</td></tr>
<tr><td>Remembers login credentials for user if approved</td><td>System</td></tr>
<tr><th colspan="2">Customer</th></tr>
<tr><th>Responsibilities</th><th>Collaborators</th></tr>
<tr><td>Browse through purchasable goods</td><td>ShoppingCart</td></tr>
<tr><td>Add purchasable goods to ShoppingCart</td><td>Item</td></tr>
<tr><td>Remove items or clear ShoppingCart</td><td>System</td></tr>
<tr><td>Complete a purchase</td><td></td></tr>
<tr><th colspan="2">Seller</th></tr>
<tr><th>Responsibilities</th><th>Collaborators</th></tr>
<tr><td>Put Items on sale</td><td>Item</td></tr>
<tr><td>Manage Items on sale</td><td>System</td></tr>
<tr><th colspan="2">Item</th></tr>
<tr><th>Responsibilities</th><th>Collaborators</th></tr>
<tr><td>Store information about the product</td><td>ShoppingCart</td></tr>
<tr><td>Store information about the availability</td><td>Seller</td></tr>
<tr><td></td><td>Customer</td></tr>
<tr><td></td><td>System</td></tr>
<tr><th colspan="2">ShoppingCart</th></tr>
<tr><th>Responsibilities</th><th>Collaborators</th></tr>
<tr><td>Store a list of Items to purchase</td><td>Item</td></tr>
<tr><td>Manage list of Items</td><td>Customer</td></tr>
<tr><th colspan="2">System</th></tr>
<tr><th>Responsibilities</th><th>Collaborators</th></tr>
<tr><td>Put items up for sale according to Seller</td><td>User</td></tr>
<tr><td>Change availability according to Seller or Customer Actions</td><td>Seller</td></tr>
<tr><td>Handles login for Users</td><td>Customer</td></tr>
<tr><td>Update balance amounts</td><td>Item</td></tr>
</table>

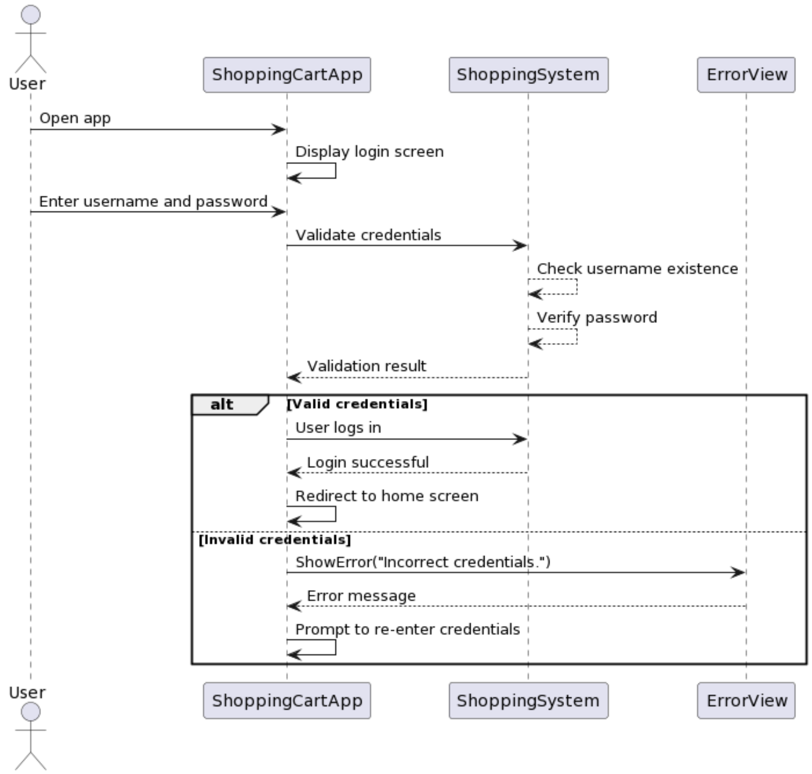
Class Diagram



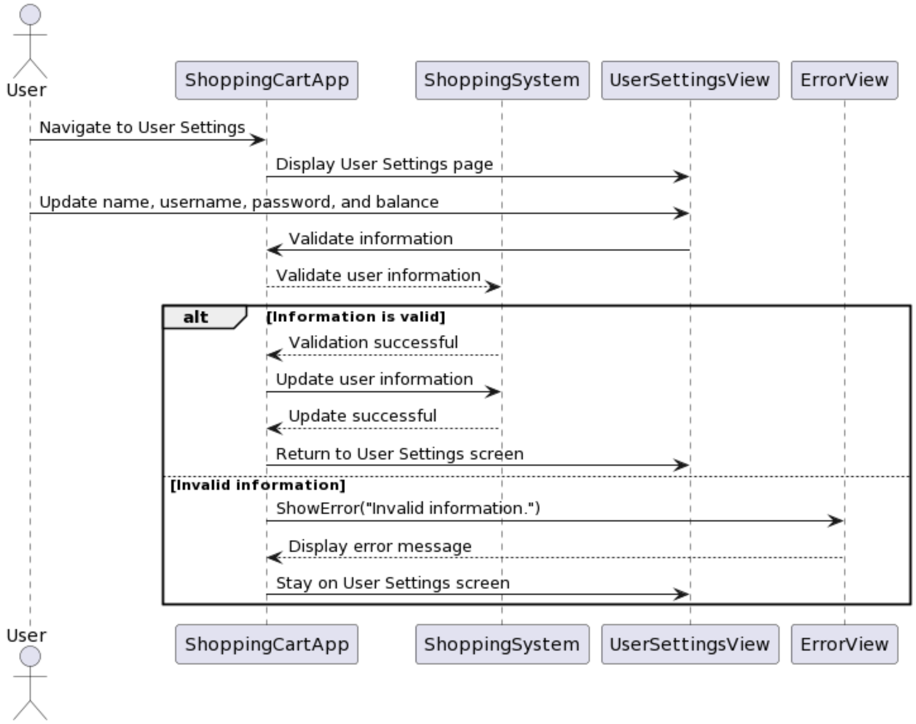
C User
username: String
password: String
balance: Float
name: String
userType: String
CheckPassword(input: String): Boolean
updateCredentials(newUsername: String, oldPassword: String, newPassword: String): void
AddFunds(amount: Float): void
RemoveFunds(amount: Float): void
ChangeName(newName: String): void
GetName(): String
GetUsername(): String
GetUserType(): String
GetBalance(): Float
C ShoppingSystem
StartApp(): void
OpenLoginScreen(): void
OpenRegisterScreen(): void
OpenSettingsScreen(): void
OpenShoppingScreen(currentCustomer: Customer): void
AddItemToCartScreen(currentCustomer: Customer, item: Item): void
OpenShoppingCartScreen(currentCustomer: Customer): void
OpenReceiptView(currentCustomer: Customer, receipt: ArrayList<String[]>, cartTotal: Float): void
OpenInventoryScreen(currentSeller: Seller): void
OpenItemEditScreen(currentSeller: Seller, itemIndex: Int): void
GetSellerFromUsername(username: String): Seller
GetCustomerFromUsername(username: String): Customer
UsernameTaken(testUsername: String): Boolean
main(Args: String[]): void
GetInstance(): ShoppingSystem
C ItemDetailsView
itemNameLabel: JLabel
itemDescriptionLabel: JLabel
sellerNameLabel: JLabel
priceTag: JLabel
stockCount: JLabel
specSheet: JScrollPane
singletonItemDetails: ItemDetailsView
ShowItem(item: Item): void
GetInstance(): ItemDetailsView
C ErrorView
errorMessage: JLabel
singletonErrorView: ErrorView
ShowError(message: String): void
showStatusSuccess(): void
GetInstance(): ErrorView
C Customer
customerCart: ShoppingCart
CompleteTransaction(): ArrayList<String[]>
Customer(username: String, password: String, name: String)
C ShoppingCart
items: ArrayList<Item>
amountPurchasing: ArrayList<Integer>
GetItem(index: Int): Item
GetPurchaseAmount(index: Int): Int
ContainsItem(item: Item): Int
SetPurchaseAmount(index: Int, amount: Int): void
RemoveItem(index: Int): void
ClearCart(): void
AddItem(newItem: Item): void
AddItem(newItem: Item, amount: Int): void
GetSize(): Int
GetTotal(): Float
C Item
name: String
description: String
price: Float
amountInStock: Int
available: Boolean
soldBy: Seller
specs: ArrayList<String>
specsName: ArrayList<String>
GetName(): String
GetDescription(): String
GetPrice(): Float
GetStock(): Int
GetSpecSize(): Int
GetAvailability(): Boolean
GetSeller(): Seller
GetSpec(index: Int): String[]
ChangeStock(amount: Int): void
SetStock(amount: Int): void
SetAvailability(available: Boolean): void
SetName(name: String): void
SetDescription(description: String): void
SetPrice(price: Float): void
SetSpecs(specs: ArrayList<String>): void
SetSpecNames(specsName: ArrayList<String>): void
equals(other: Object): Boolean
hashCode(): Int
C Seller
inventory: ArrayList<Item>
SellItem(product: Item): void
SellItem(product: Item, stock: Int): void
UpdateItem(updatedProduct: Item, index: Int): void
GetInventorySize(): Int
GetItemInInventory(index: Int): Item
RemoveFromSale(index: Int): void
ChangeStock(index: Int, amount: Int): void
ChangeAvailability(index: Int, available: Boolean): void
Seller(username: String, password: String, name: String)

Sequence Diagrams

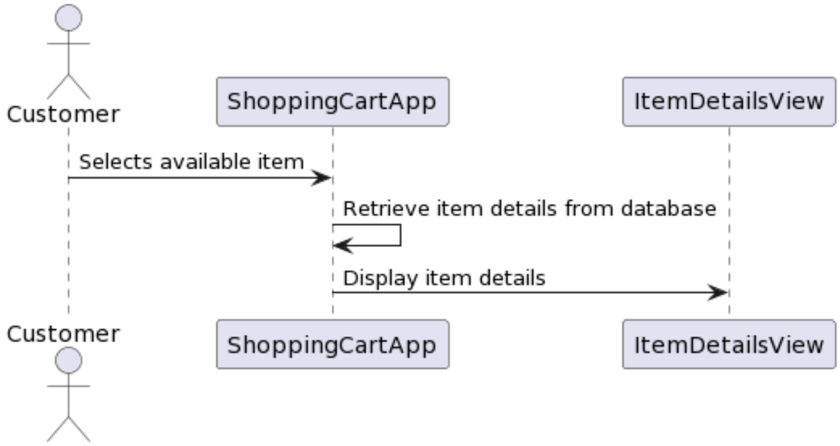
User Logs In:



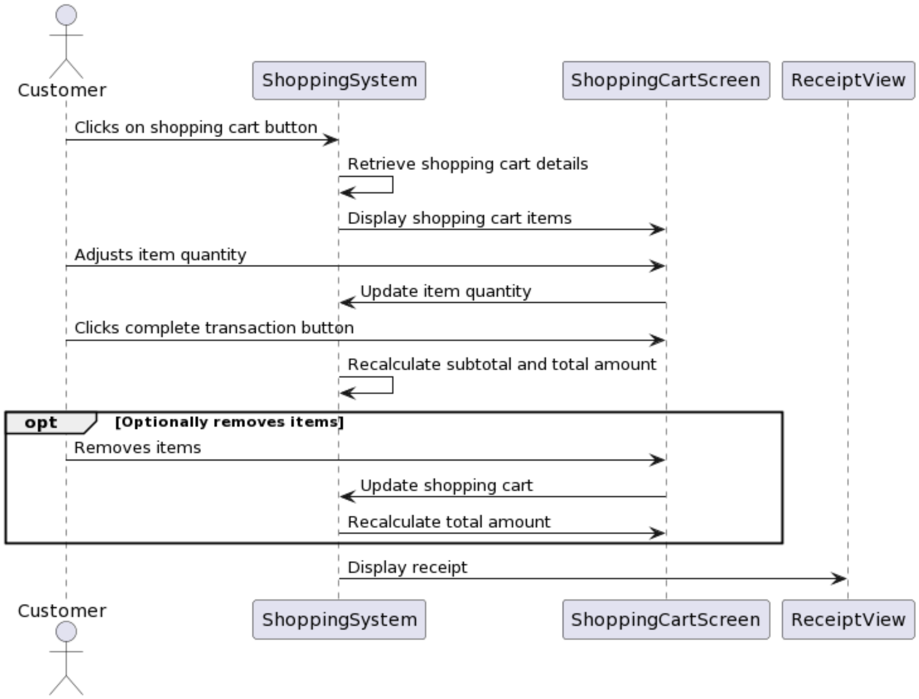
User Changes/Updates their Login Credentials and Balance



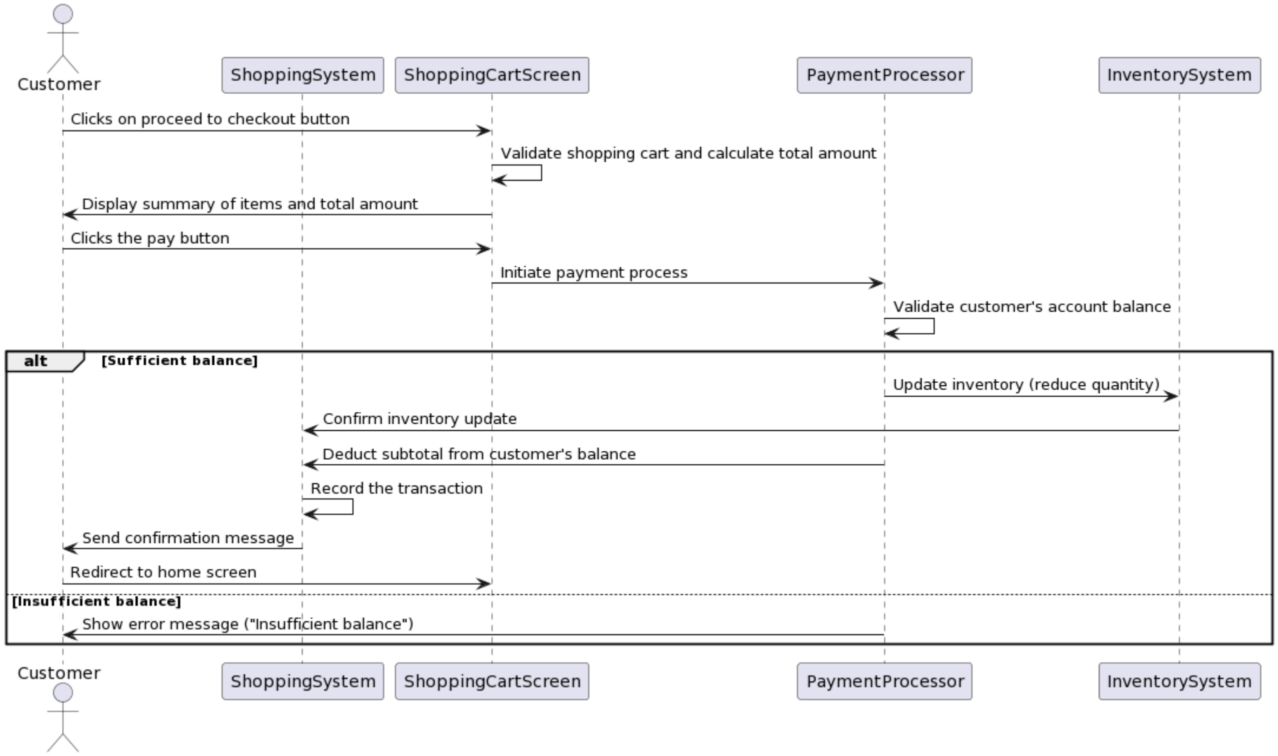
Customer Reviews Product Details



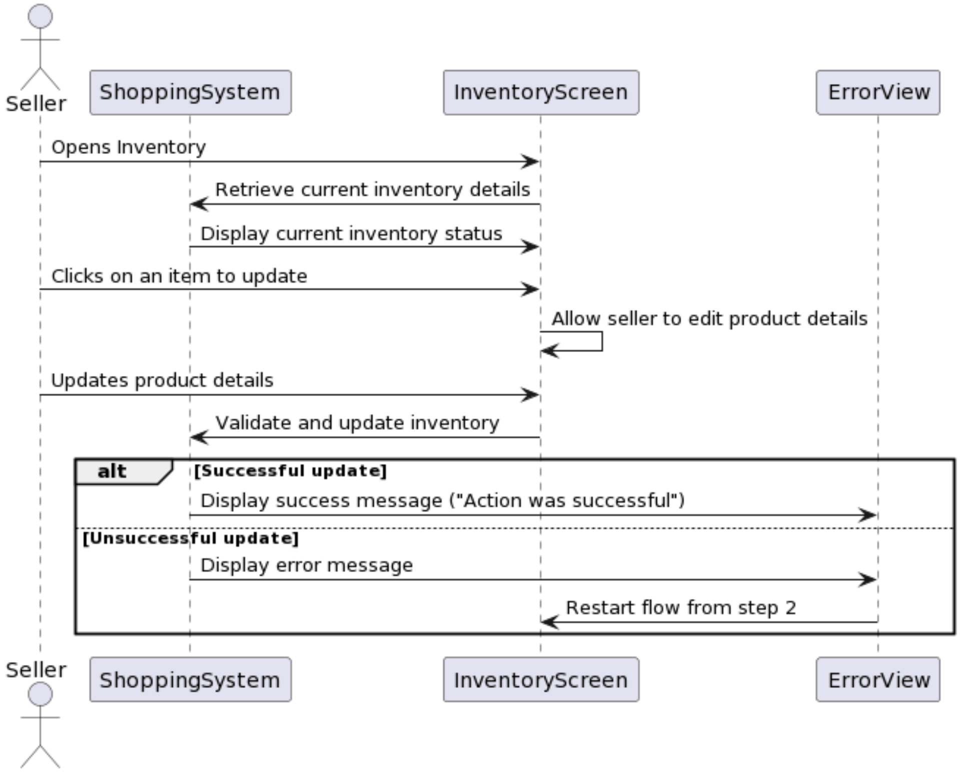
Customer Reviews/Updates Shopping Cart



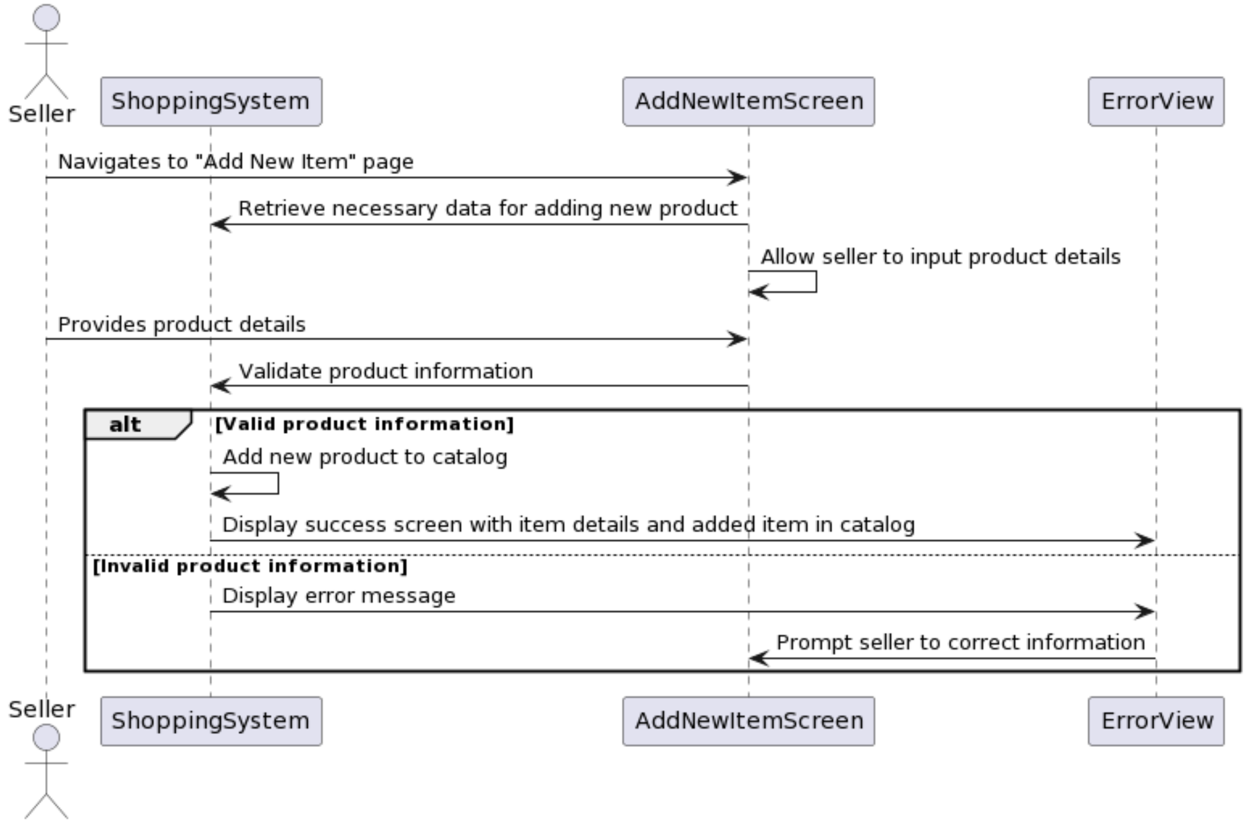
Customer Checks Out


Customer
ShoppingSystem
ShoppingCartScreen
PaymentProcessor
InventorySystem
Clicks on proceed to checkout button
Validate shopping cart and calculate total amount
Display summary of items and total amount
Clicks the pay button
Initiate payment process
Validate customer's account balance
alt
[Sufficient balance]
Update inventory (reduce quantity)
Confirm inventory update
Deduct subtotal from customer's balance
Record the transaction
Send confirmation message
Redirect to home screen
[Insufficient balance]
Show error message ("Insufficient balance")
Customer
ShoppingSystem
ShoppingCartScreen
PaymentProcessor
InventorySystem

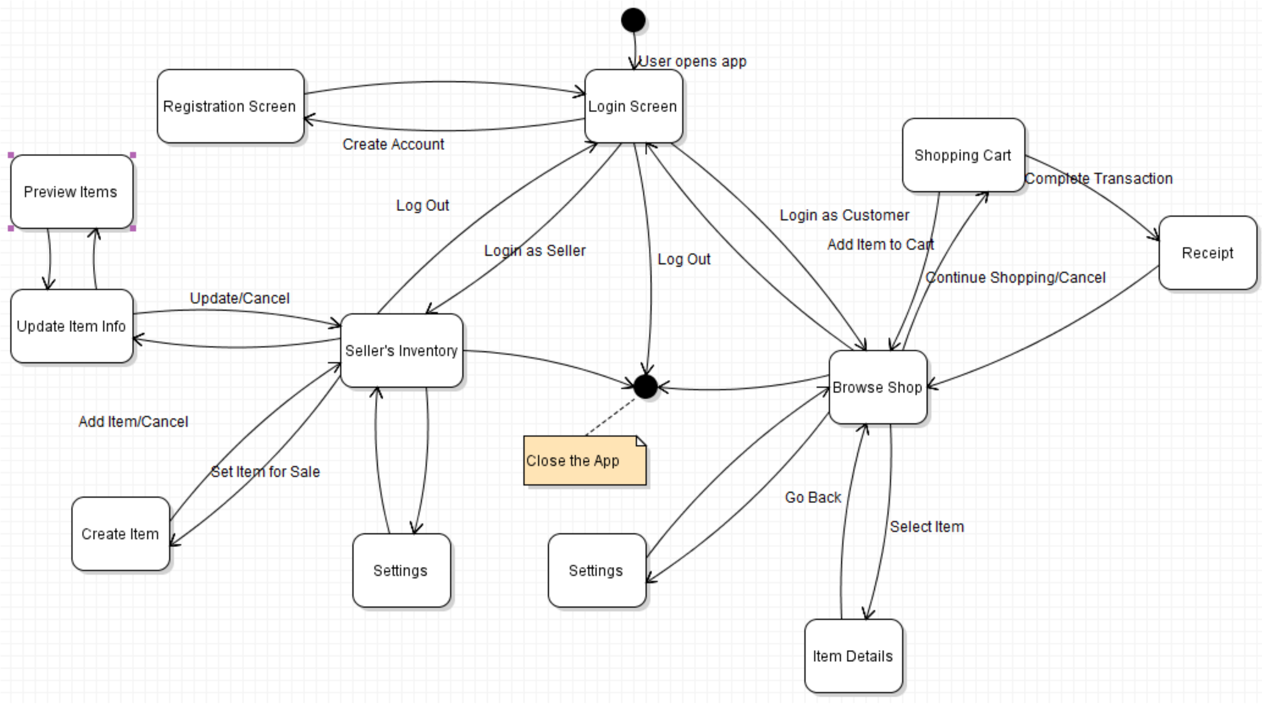
Seller Reviews/Updates Inventory



Seller Adds New Product


Seller
ShoppingSystem
AddNewItemScreen
ErrorView
Navigates to "Add New Item" page
Retrieve necessary data for adding new product
Allow seller to input product details
Provides product details
Validate product information
alt
[Valid product information]
Add new product to catalog
Display success screen with item details and added item in catalog
[Invalid product information]
Display error message
Prompt seller to correct information
Seller
ShoppingSystem
AddNewItemScreen
ErrorView

State Diagram



Java Code: GUI

ErrorView.java

```java
package cop4331.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * The ErrorView class represents a view for displaying error messages in a
graphical user interface.
 * This class extends JFrame and provides methods to show error messages and
success statuses.
 * It follows the Singleton design pattern to maintain a single instance
throughout the application.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class ErrorView extends JFrame {
    /**
     * Private constructor to prevent external instantiation.
     */
    private ErrorView() {
        super();
        errorMessage = new JLabel();
        setLayout(new BorderLayout());
        add(errorMessage, BorderLayout.CENTER);
        JButton OkButton = new JButton("OK");
        OkButton.addActionListener((ActionEvent e) -> {
this.setVisible(false); });
        add(OkButton, BorderLayout.PAGE_END);
        setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        OkButton.addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {}

            @Override
            public void keyPressed(KeyEvent theKey) {
                if (theKey.getExtendedKeyCode() == KeyEvent.VK_ENTER) {
                    OkButton.doClick();
                }
            }

            @Override
            public void keyReleased(KeyEvent e) {}
        });
    }
    /**
     * Displays an error message in the ErrorView.
     * @param message The error message to be displayed.
     */
    public void ShowError(String message) {
        setTitle("An error has occurred!");
        errorMessage.setText(message);
        errorMessage.repaint();
        pack();
        setVisible(true);
    }

    /**
     * Displays a success status message in the ErrorView.
     */
    public void showStatusSuccess() {
        setTitle("Success!");
        errorMessage.setText("Action was successful!");
        errorMessage.repaint();
        pack();
        setVisible(true);
    }

    /**
     * Retrieves the singleton instance of ErrorView.
     * @return The singleton instance of ErrorView.
     */
    public static ErrorView GetInstance() {
        return singletonErrorView;
    }

    // Private fields

    /** JLabel to display error messages */
    private final JLabel errorMessage;

    /** Singleton instance of ErrorView */
    private final static ErrorView singletonErrorView = new ErrorView();
}
```

ItemDetailsView.java

```java
package cop4331.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import cop4331.models.Item;

/**
 * The ItemDetailsView class represents a view for displaying details of an
item in a graphical user interface.
 * This class extends JFrame and follows the Singleton design pattern to
maintain a single instance throughout the application.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class ItemDetailsView extends JFrame {
    /**
     * Private constructor to prevent external instantiation.
     */
    private ItemDetailsView() {
        super();
        setLayout(new BorderLayout());
        JPanel superPanel = new JPanel();
        superPanel.setLayout(new BoxLayout(superPanel, BoxLayout.PAGE_AXIS));
        JPanel namePanel = new JPanel();
        namePanel.setLayout(new FlowLayout());
        namePanel.add(itemNameLabel);
        JPanel descPanel = new JPanel();
        descPanel.setLayout(new BoxLayout(descPanel, BoxLayout.LINE_AXIS));
        descPanel.add(itemDescriptionLabel);
        JPanel sellerPanel = new JPanel();
        sellerPanel.setLayout(new BoxLayout(sellerPanel,
BoxLayout.LINE_AXIS));
        sellerPanel.add(sellerNameLabel);
        superPanel.add(namePanel);
        superPanel.add(descPanel);
        superPanel.add(sellerPanel);
        JPanel valuePanel = new JPanel();
        valuePanel.setLayout(new BoxLayout(valuePanel, BoxLayout.LINE_AXIS));
        valuePanel.add(priceTag);
        JPanel spacerPanel = new JPanel();
        spacerPanel.setSize(10, 0);
        valuePanel.add(spacerPanel);
        valuePanel.add(stockCount);
        superPanel.add(valuePanel);
        superPanel.add(specSheet);
        add(superPanel, BorderLayout.CENTER);
        JButton OkButton = new JButton("Done");
        OkButton.addActionListener((ActionEvent e) -> {
this.setVisible(false); });
        add(OkButton, BorderLayout.PAGE_END);
        setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        OkButton.addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {}

            @Override
            public void keyPressed(KeyEvent theKey) {
                if (theKey.getExtendedKeyCode() == KeyEvent.VK_ENTER) {
                    OkButton.doClick();
                }
            }

            @Override
            public void keyReleased(KeyEvent e) {}
        });
    }

    /**
     * Displays the details of the given Item in the ItemDetailsView.
     * @param item The Item object whose details are to be displayed.
     */
    public void ShowItem(Item item) {
        setTitle("Item Details");
        itemNameLabel.setText(item.GetName());
        itemDescriptionLabel.setText(item.GetDescription());
        sellerNameLabel.setText(item.GetSeller().GetName());
        priceTag.setText("Price: " + String.valueOf(item.GetPrice()) + "$");
        stockCount.setText("Current Stock: " +
String.valueOf(item.GetStock()));
        JPanel specSheetPanel = new JPanel();
        specSheetPanel.setLayout(new BoxLayout(specSheetPanel,
BoxLayout.PAGE_AXIS));
        for (int i = 0; i < item.GetSpecSize(); i++) {
            JPanel specPanel = new JPanel();
            specPanel.setLayout(new BoxLayout(specPanel,
BoxLayout.LINE_AXIS));
            String[] itemSpec = item.GetSpec(i);
            JLabel specName = new JLabel(itemSpec[0]);
            JLabel specDesc = new JLabel(itemSpec[1]);
            JPanel spacerPanel = new JPanel();
            spacerPanel.setSize(10, 0);
            specPanel.add(specName);
            specPanel.add(spacerPanel);
            specPanel.add(specDesc);
            if (i % 2 == 1) {
                specPanel.setBackground(Color.LIGHT_GRAY);
                spacerPanel.setBackground(Color.LIGHT_GRAY);
            } else {
                specPanel.setBackground(Color.WHITE);
                spacerPanel.setBackground(Color.WHITE);
            }
            specSheetPanel.add(specPanel);
        }
        specSheet.setViewportView(specSheetPanel);

specSheet.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER
);

specSheet.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED
);
        repaint();
        pack();
        setVisible(true);
    }

    /**
     * Retrieves the singleton instance of ItemDetailsView.
     * @return The singleton instance of ItemDetailsView.
     */
    public static ItemDetailsView GetInstance() {
        return singletonItemDetails;
    }

    // Private fields

    /** JLabel to display item name */
    private final JLabel itemNameLabel = new JLabel();

    /** JLabel to display item description */
    private final JLabel itemDescriptionLabel = new JLabel();

    /** JLabel to display seller name */
    private final JLabel sellerNameLabel = new JLabel();

    /** JLabel to display item price */
    private final JLabel priceTag = new JLabel();

    /** JLabel to display stock count */
    private final JLabel stockCount = new JLabel();

    /** JScrollPane to contain item specifications */
    private final JScrollPane specSheet = new JScrollPane();

    /** Singleton instance of ItemDetailsView */
    private final static ItemDetailsView singletonItemDetails = new
ItemDetailsView();
}
```

ShoppingSystem.java

```java
package cop4331.gui;

import cop4331.models.*;
import java.util.ArrayList;
import java.util.NoSuchElementException;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import javax.swing.table.*;

/**
 * The ShoppingSystem class manages the main functionality of the shopping
application.
 * It handles user interactions, data storage, and screen transitions.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class ShoppingSystem {
    /**
     * Private default constructor for the ShoppingSystem singleton.
     */
    private ShoppingSystem(){}

    /**
     * Starts the shopping application by setting up the main view, loading
data from files,
     * and displaying the login screen.
     */
    public void StartApp()
    {
        GetDataFromFiles();
        mainView.setTitle("Shopping Cart Application");
        mainView.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        mainView.addWindowListener(new WindowListener(){
            @Override
            public void windowOpened(WindowEvent e) {}
            @Override
            public void windowClosing(WindowEvent e) {
                try
                    (FileOutputStream writeUserData = new
FileOutputStream("userData.txt");
                    ObjectOutputStream userWriter = new
ObjectOutputStream(writeUserData))
                {
                    ArrayList<User> usersList = new ArrayList<User>();
                    for(int i=0;i<sellersList.size();i++)
                    {usersList.add((User) sellersList.get(i));}
                    for(int i=0;i<customersList.size();i++)
                    {usersList.add((User) customersList.get(i));}
                    userWriter.writeObject(usersList);
                    writeUserData.close();
                    userWriter.close();
                } catch (IOException ex) {System.out.println("Failed to save
data.");}
                System.exit(0);
            }
            @Override
            public void windowClosed(WindowEvent e) {}
            @Override
            public void windowIconified(WindowEvent e) {}
            @Override
            public void windowDeiconified(WindowEvent e) {}
            @Override
            public void windowActivated(WindowEvent e) {}
            @Override
            public void windowDeactivated(WindowEvent e) {}
        });
        mainView.setVisible(true);
        mainView.add(container);
        OpenLoginScreen();
    }
    /**
     * Loads user data from files when the application starts.
     */
    private void GetDataFromFiles()
    {
        sellersList = new ArrayList<Seller>();
        customersList = new ArrayList<Customer>();
        try
            (FileInputStream userData = new FileInputStream("userData.txt");
            ObjectInputStream userReader = new ObjectInputStream(userData))
        {
            ArrayList<User> usersList = (ArrayList<User>)
userReader.readObject();
            userReader.close();
            userData.close();
            for(int i=0;i<usersList.size();i++)
            {
                User tempUser = usersList.get(i);
                if(tempUser.GetUserType().equals("Seller"))
                {
                    Seller currentSeller = (Seller) tempUser;
                    sellersList.add(currentSeller);
                    for(int j=0;j<currentSeller.GetInventorySize();j++)

{currentSeller.GetItemInInventory(j).SetSeller(currentSeller);}
                }
                else if(tempUser.GetUserType().equals("Customer"))
                {customersList.add((Customer) tempUser);}
            }
        }
        catch(FileNotFoundException e)
        {
            File userDataFile = new File("userData.txt");
            System.out.println("File not found, created file
"+userDataFile.getName());
        }
        catch(IOException | ClassNotFoundException e){}
    }

    /**
     * Resets the main view by removing its components.
     */
    private void ResetMainView()
    {
        container.removeAll();
        container.revalidate();
        container.repaint();
    }

    /**
     * Opens the login screen where users can log in to their accounts.
     */
    public void OpenLoginScreen()
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container, BoxLayout.PAGE_AXIS));
        // Username Field
        JPanel usernameContainer = new JPanel();
        usernameContainer.setLayout(new BoxLayout(usernameContainer,
BoxLayout.LINE_AXIS));
        JLabel usernameLabel = new JLabel("Username: ");
        JTextField usernameTextField = new JTextField(10);
        usernameContainer.add(usernameLabel);
        usernameContainer.add(usernameTextField);
        container.add(usernameContainer);
        // Password Field
        JPanel passwordContainer = new JPanel();
        passwordContainer.setLayout(new BoxLayout(passwordContainer,
BoxLayout.LINE_AXIS));
        JLabel passwordLabel = new JLabel("Password: ");
        JTextField passwordTextField = new JTextField(10);
        passwordContainer.add(passwordLabel);
        passwordContainer.add(passwordTextField);
        container.add(passwordContainer);
        // Buttons
        JButton registerInstead = new JButton("Don't have an account?
Register here.");
        container.add(registerInstead);
        registerInstead.setAlignmentX(JButton.CENTER_ALIGNMENT);
        JButton loginButton = new JButton("Log in");
        container.add(loginButton);
        loginButton.setAlignmentX(JButton.CENTER_ALIGNMENT);
        loginButton.addActionListener((ActionEvent e)->{

if(usernameTextField.getText().equals("")||passwordTextField.getText().equals
(""))
            {errorView.ShowError("One or more of the fields are empty.");}
            else
            {
                try{
                    Seller foundAccount =
GetSellerFromUsername(usernameTextField.getText());

if(foundAccount.CheckPassword(passwordTextField.getText()))
                    {
                        currentUser = foundAccount;
                        OpenInventoryScreen(foundAccount);
                    }else{errorView.ShowError("Incorrect credentials.");}
                }catch(NoSuchElementException ex1){
                    try{
                        Customer foundAccount =
GetCustomerFromUsername(usernameTextField.getText());

if(foundAccount.CheckPassword(passwordTextField.getText()))
                        {
                            currentUser = foundAccount;
                            OpenShoppingScreen(foundAccount);
                        }else{errorView.ShowError("Incorrect credentials.");}
                    }catch(NoSuchElementException ex2)
                    {errorView.ShowError("Incorrect credentials.");}
                }
            }
        });
        // Adding an ActionListener to registerInstead button
registerInstead.addActionListener((ActionEvent e) -> {
    // Calling the method to open the register screen
    OpenRegisterScreen();
});

// Creating a KeyListener for login functionality
KeyListener loginOnEnter = new KeyListener() {
    @Override
    public void keyTyped(KeyEvent e) {
        // Not handling keyTyped event, leaving it empty
    }

    @Override
    public void keyPressed(KeyEvent theKey) {
        // Checking if the pressed key is the 'Enter' key
        if (theKey.getExtendedKeyCode() == KeyEvent.VK_ENTER) {
            // If 'Enter' is pressed, simulate a click on the loginButton
            loginButton.doClick();
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {
        // Not handling keyReleased event, leaving it empty
    }
};

// Adding the loginOnEnter KeyListener to the username and password text
fields
usernameTextField.addKeyListener(loginOnEnter);
passwordTextField.addKeyListener(loginOnEnter);
        //mainView.add(container);
        mainView.pack();
    }

    /**
     * Opens the registration screen where new users can create accounts.
     */
    public void OpenRegisterScreen()
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container, BoxLayout.PAGE_AXIS));
        // Username Field
        JPanel usernameContainer = new JPanel();
        usernameContainer.setLayout(new BoxLayout(usernameContainer,
BoxLayout.LINE_AXIS));
        JLabel usernameLabel = new JLabel("Username: ");
        JTextField usernameTextField = new JTextField(10);
        usernameContainer.add(usernameLabel);
        usernameContainer.add(usernameTextField);
        container.add(usernameContainer);
        // Password Field
        JPanel passwordContainer = new JPanel();
        passwordContainer.setLayout(new BoxLayout(passwordContainer,
BoxLayout.LINE_AXIS));
        JLabel passwordLabel = new JLabel("Password: ");
        JTextField passwordTextField = new JTextField(10);
        passwordContainer.add(passwordLabel);
        passwordContainer.add(passwordTextField);
        container.add(passwordContainer);
        // Name Field
        JPanel nameContainer = new JPanel();
        nameContainer.setLayout(new BoxLayout(nameContainer,
BoxLayout.LINE_AXIS));
        JLabel nameLabel = new JLabel("Name: ");
        JTextField nameTextField = new JTextField(10);
        nameContainer.add(nameLabel);
        nameContainer.add(nameTextField);
        container.add(nameContainer);
        // Account Type
        JPanel selectionContainer = new JPanel();
        selectionContainer.setLayout(new BoxLayout(selectionContainer,
BoxLayout.LINE_AXIS));
        ButtonGroup accountTypeSelection = new ButtonGroup();
        JRadioButton customerSelection = new JRadioButton("Customer");
        JRadioButton sellerSelection = new JRadioButton("Seller");
        selectionContainer.add(customerSelection);
        selectionContainer.add(sellerSelection);
        accountTypeSelection.add(customerSelection);
        accountTypeSelection.add(sellerSelection);
        container.add(selectionContainer);
        customerSelection.setSelected(true);
        // Buttons
        JButton loginInstead = new JButton("Already have an account? Log in
here.");
        container.add(loginInstead);
        loginInstead.setAlignmentX(JButton.CENTER_ALIGNMENT);
        JButton registerButton = new JButton("Register");
        container.add(registerButton);
        registerButton.setAlignmentX(JButton.CENTER_ALIGNMENT);
        registerButton.addActionListener((ActionEvent e)->{

if(usernameTextField.getText().equals("")||passwordTextField.getText().equals
("")||nameTextField.getText().equals(""))
            {errorView.ShowError("One or more of the fields are empty.");}
            else
            {
                try{
                    GetSellerFromUsername(usernameTextField.getText());
                    errorView.ShowError("This username is already taken.");
                }catch(NoSuchElementException ex1){
                    try{
                        GetCustomerFromUsername(usernameTextField.getText());
                        errorView.ShowError("This username is already
taken.");
                    }catch(NoSuchElementException ex2)
                    {
                        if(customerSelection.isSelected())
                        {
                            Customer currentCustomer = new
Customer(usernameTextField.getText(), passwordTextField.getText(),
nameTextField.getText());
                            customersList.add(currentCustomer);
                            currentUser = currentCustomer;
                            OpenShoppingScreen(currentCustomer);
                        }else if(sellerSelection.isSelected()){
                            Seller currentSeller = new
Seller(usernameTextField.getText(), passwordTextField.getText(),
nameTextField.getText());
                            sellersList.add(currentSeller);
                            currentUser = currentSeller;
                            OpenInventoryScreen(currentSeller);
                        }else{errorView.ShowError("An account type hasn't
been selected.");}
```

```
                    }
                }
            }
        });
        // Adding an ActionListener to loginInstead button
loginInstead.addActionListener((ActionEvent e) -> {
    // Calling the method to open the login screen
    OpenLoginScreen();
});

// Creating a KeyListener for registration functionality
KeyListener registerOnEnter = new KeyListener() {
    @Override
    public void keyTyped(KeyEvent e) {
        // Not handling keyTyped event, leaving it empty
    }

    @Override
    public void keyPressed(KeyEvent theKey) {
        // Checking if the pressed key is the 'Enter' key
        if (theKey.getExtendedKeyCode() == KeyEvent.VK_ENTER) {
            // If 'Enter' is pressed, simulate a click on the registerButton
            registerButton.doClick();
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {
        // Not handling keyReleased event, leaving it empty
    }
};

// Adding the registerOnEnter KeyListener to the username, password, and name
text fields
usernameTextField.addKeyListener(registerOnEnter);
passwordTextField.addKeyListener(registerOnEnter);
nameTextField.addKeyListener(registerOnEnter);

// Packing the mainView (assuming mainView is a Swing container/window)
mainView.pack();


    }

    /**
     * Opens the settings screen where users can manage their account
settings.
     */
    public void OpenSettingsScreen()
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container, BoxLayout.PAGE_AXIS));
        // Top Bar
        JPanel titleBar = new JPanel();
        titleBar.setLayout(new BoxLayout(titleBar, BoxLayout.LINE_AXIS));
        JButton goBack = new JButton("Go Back");
        JLabel settingsLabel = new JLabel("Settings");
        JButton logoutButton = new JButton("Log out");
        titleBar.add(goBack);
        titleBar.add(settingsLabel);
        titleBar.add(logoutButton);
        goBack.addActionListener((ActionEvent e)->{
            switch (currentUser.GetUserType()) {
                case "Seller" -> OpenInventoryScreen((Seller)currentUser);
                case "Customer" -> OpenShoppingScreen((Customer)currentUser);
                default -> errorView.ShowError("Something went wrong! Unable
to get user's account type, please close the app or log out.");
            }
        });
        logoutButton.addActionListener((ActionEvent e)->{
            currentUser = null;
            OpenLoginScreen();
        });
        container.add(titleBar);
        titleBar.setAlignmentX(JPanel.CENTER_ALIGNMENT);
        // Username
        JPanel usernameBar = new JPanel();
        usernameBar.setLayout(new
BoxLayout(usernameBar,BoxLayout.LINE_AXIS));
        JLabel usernameLabel = new JLabel("Username: ");
        JTextField usernameTextField = new
JTextField(currentUser.GetUsername());
        usernameBar.add(usernameLabel);
        usernameBar.add(usernameTextField);
        container.add(usernameBar);
        // Name
        JPanel nameBar = new JPanel();
        nameBar.setLayout(new BoxLayout(nameBar,BoxLayout.LINE_AXIS));
        JLabel nameLabel = new JLabel("Name: ");
        JTextField nameTextField = new JTextField(currentUser.GetName());
        nameBar.add(nameLabel);
        nameBar.add(nameTextField);
        container.add(nameBar);
        // Password
        JPanel newPasswordBar = new JPanel();
        newPasswordBar.setLayout(new
BoxLayout(newPasswordBar,BoxLayout.LINE_AXIS));
        JLabel newPasswordLabel = new JLabel("New Password: ");
        JTextField newPasswordTextField = new JTextField();
        newPasswordBar.add(newPasswordLabel);
        newPasswordBar.add(newPasswordTextField);
        JLabel newPasswordNote1 = new JLabel("*You only need to enter your
current password when changing your password.");
        JLabel newPasswordNote2 = new JLabel("Leave the above field blank and
you dont have to enter your current password.");
        container.add(newPasswordBar);
        container.add(newPasswordNote1);
        container.add(newPasswordNote2);
        // Current Password
        JPanel oldPasswordBar = new JPanel();
        oldPasswordBar.setLayout(new
BoxLayout(oldPasswordBar,BoxLayout.LINE_AXIS));
        JLabel oldPasswordLabel = new JLabel("Current Password: ");
        JTextField oldPasswordTextField = new JTextField();
        oldPasswordBar.add(oldPasswordLabel);
        oldPasswordBar.add(oldPasswordTextField);
        container.add(oldPasswordBar);
        // Add Funds
        JPanel addFundsBar = new JPanel();
        addFundsBar.setLayout(new
BoxLayout(addFundsBar,BoxLayout.LINE_AXIS));
        JLabel fundsLabel = new JLabel("Add Funds: ");
        JTextField fundsTextField = new JTextField();
        addFundsBar.add(fundsLabel);
        addFundsBar.add(fundsTextField);
        container.add(addFundsBar);
        // Save Button
        JButton saveSettings = new JButton("Save");
        container.add(saveSettings);
        saveSettings.setAlignmentX(JButton.CENTER_ALIGNMENT);
        saveSettings.addActionListener((ActionEvent e) -> {

if(usernameTextField.getText().equals("")||nameTextField.getText().equals("")
)
            {
                errorView.ShowError("One or more of the required fields are
empty.");
                return;
            }
            try{
                Seller foundUser =
GetSellerFromUsername(usernameTextField.getText());
                if(foundUser.GetUsername().equals(currentUser.GetUsername()))
                {throw new NoSuchElementException();}
                errorView.ShowError("This username is already taken.");
            }catch(NoSuchElementException ex1){
                try{
                    Customer foundUser =
GetCustomerFromUsername(usernameTextField.getText());

if(foundUser.GetUsername().equals(currentUser.GetUsername()))
                    {throw new NoSuchElementException();}
                    errorView.ShowError("This username is already taken.");
                }catch(NoSuchElementException ex2)
                {
                    if(!newPasswordTextField.getText().equals(""))
                    {
                        try
{currentUser.ChangePassword(oldPasswordTextField.getText(),
newPasswordTextField.getText());}
                        catch(Exception ex)
                        {
                            errorView.ShowError("You have attempted to change
your password but entered your current password incorrectly. Please try
again.");
                            return;
                        }
                    }
                    currentUser.ChangeName(nameTextField.getText());
                    currentUser.ChangeUsername(usernameTextField.getText());
                    oldPasswordTextField.setText("");
                    newPasswordTextField.setText("");
                    if(!fundsTextField.getText().equals(""))
                    {
                        try
                        {
                            float inputFunds =
Float.parseFloat(fundsTextField.getText());
                            currentUser.AddFunds(inputFunds);
                            fundsTextField.setText("");
                            errorView.showStatusSuccess();
                        }
                        catch(NumberFormatException ex3)
                        {errorView.ShowError("Add funds field has been
entered incorrectly. Other changes to account have been saved.");}
                        catch (Exception ex)
                        {errorView.ShowError("Can not add negative funds to
balance. Other changes to account have been saced.");}
                    }
                    else
                    {errorView.showStatusSuccess();}
                }
            }
        });
        // Creating a KeyListener for saving settings functionality
KeyListener saveOnEnter = new KeyListener() {
    @Override
    public void keyTyped(KeyEvent e) {
        // Not handling keyTyped event, leaving it empty
    }

    @Override
    public void keyPressed(KeyEvent theKey) {
        // Checking if the pressed key is the 'Enter' key
        if (theKey.getExtendedKeyCode() == KeyEvent.VK_ENTER) {
            // If 'Enter' is pressed, simulate a click on the saveSettings
button
            saveSettings.doClick();
        }
    }
    @Override
    public void keyReleased(KeyEvent e) {
        // Not handling keyReleased event, leaving it empty
    }
};

// Adding the saveOnEnter KeyListener to specific text fields
usernameTextField.addKeyListener(saveOnEnter);
nameTextField.addKeyListener(saveOnEnter);
newPasswordTextField.addKeyListener(saveOnEnter);
oldPasswordTextField.addKeyListener(saveOnEnter);

// Packing the mainView (assuming mainView is a Swing container/window)
mainView.pack();

    }

    /**
     * Opens the shopping screen for a specific customer, displaying
available items for purchase.
     * This method sets up the GUI to show available items for the customer
to browse and buy.
     * It populates the screen with a list of items from sellers and enables
the user to interact
     * with each item to potentially add them to the shopping cart.
     * @param currentCustomer The customer currently logged into the shopping
system.
     */
    public void OpenShoppingScreen(Customer currentCustomer)
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container, BoxLayout.PAGE_AXIS));
        // Title Bar
        JPanel titleBar = new JPanel();
        JLabel titleLabel = new JLabel("Shopping Menu");
        JButton logoutButton = new JButton("Log out");
        JButton settingsButton = new JButton("User Settings");
        JButton cartButton = new JButton("Shopping Cart
("+String.valueOf(currentCustomer.customerCart.GetSize())+")");
        JLabel balanceLabel = new JLabel("Balance:
"+String.valueOf(currentCustomer.GetBalance()));
        titleBar.add(cartButton);
        titleBar.add(titleLabel);
        titleBar.add(logoutButton);
        titleBar.add(settingsButton);
        titleBar.add(balanceLabel);
        logoutButton.addActionListener((ActionEvent e)->{
            currentUser = null;
            OpenLoginScreen();
        });
        settingsButton.addActionListener((ActionEvent e)->{
            OpenSettingsScreen();
        });
        cartButton.addActionListener((ActionEvent e)->{
            OpenShoppingCartScreen(currentCustomer);
        });
        container.add(titleBar);
        // Inventory List
        JPanel shoppingPanel = new JPanel();
        shoppingPanel.setLayout(new
BoxLayout(shoppingPanel,BoxLayout.PAGE_AXIS));
        ArrayList<Item> itemsListed = new ArrayList<Item>();
        int totalIndex = 0;
        for(int j=0;j<sellersList.size();j++)
        {
            Seller currentSeller = sellersList.get(j);
            for(int i=0;i<currentSeller.GetInventorySize();i++)
            {
                Item currentItem = currentSeller.GetItemInInventory(i);
                if (currentItem.GetAvailability())
                {
                    int myIndex = totalIndex;
                    totalIndex++;
                    itemsListed.add(currentItem);
                    JPanel itemSuperPanel = new JPanel();
                    itemSuperPanel.setLayout(new BoxLayout(itemSuperPanel,
BoxLayout.LINE_AXIS));
                    JPanel itemLeftPanel = new JPanel();
                    itemLeftPanel.setLayout(new BoxLayout(itemLeftPanel,
BoxLayout.PAGE_AXIS));
                    JPanel itemRightPanel = new JPanel();
                    itemRightPanel.setLayout(new BoxLayout(itemRightPanel,
BoxLayout.PAGE_AXIS));
                    itemLeftPanel.add(new JLabel(currentItem.GetName()));
                    itemLeftPanel.add(new JLabel("Seller:
"+currentItem.GetSeller().GetName()));
                    itemRightPanel.add(new JLabel("Price:
"+String.valueOf(currentItem.GetPrice())+"$"));
                    itemRightPanel.add(new JLabel("Stock:
"+String.valueOf(currentItem.GetStock())));
                    shoppingPanel.add(itemSuperPanel);
                    itemSuperPanel.add(itemLeftPanel);
                    JPanel spacePanel = new JPanel();
                    spacePanel.setSize(10, 0);
                    itemSuperPanel.add(spacePanel);
                    itemSuperPanel.add(itemRightPanel);
                    itemSuperPanel.addMouseListener(new MouseListener(){
                        @Override
                        public void mouseClicked(MouseEvent e)
                        {AddItemToCartScreen(currentCustomer,
itemsListed.get(myIndex));}
                        @Override
                        public void mousePressed(MouseEvent e) {}
                        @Override
                        public void mouseReleased(MouseEvent e) {}
                        @Override
                        public void mouseEntered(MouseEvent e) {}
                        @Override
                        public void mouseExited(MouseEvent e) {}
                    });
                    if(myIndex%2==1){
                        itemSuperPanel.setBackground(Color.LIGHT_GRAY);
                        itemLeftPanel.setBackground(Color.LIGHT_GRAY);
                        spacePanel.setBackground(Color.LIGHT_GRAY);
                        itemRightPanel.setBackground(Color.LIGHT_GRAY);
                    }
                    else{
                        itemSuperPanel.setBackground(Color.WHITE);
                        itemLeftPanel.setBackground(Color.WHITE);
                        spacePanel.setBackground(Color.WHITE);
                        itemRightPanel.setBackground(Color.WHITE);
                    }
                }
            }
        }
        JScrollPane shoppingView = new JScrollPane(shoppingPanel);

shoppingView.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NE
VER);

shoppingView.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEE
DED);
        container.add(shoppingView);
        mainView.pack();
    }

    /**
     * Displays a screen for adding an item to the customer's cart.
     * This method shows a GUI screen where the user can view details of a
specific item
     * and add it to their shopping cart with a specified quantity.
     * @param currentCustomer The customer currently logged into the shopping
system.
     * @param item            The item the user intends to add to their
shopping cart.
     */
    public void AddItemToCartScreen(Customer currentCustomer, Item item)
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container, BoxLayout.PAGE_AXIS));
        JLabel itemName = new JLabel(item.GetName());
        JPanel namePanel = new JPanel();
        namePanel.setLayout(new FlowLayout());
        namePanel.add(itemName);
        JLabel itemDescription = new JLabel(item.GetDescription());
        JPanel descPanel = new JPanel();
        descPanel.setLayout(new FlowLayout());
        descPanel.add(itemDescription);
        JPanel stockPanel = new JPanel();
        JLabel stockLabel = new JLabel("How many to add to cart?");
        JPanel stockSpacer = new JPanel();
        stockSpacer.setSize(10,0);
        JTextField stockTextField = new JTextField("1");
        stockTextField.setColumns(3);
        int itemCartIndex = currentCustomer.customerCart.ContainsItem(item);
        if(itemCartIndex>=0)
        {

stockTextField.setText(String.valueOf(currentCustomer.customerCart.GetPurchas
eAmount(itemCartIndex)));
            stockLabel.setText("Item is in the cart. Would you like to change
the amount?");
        }
        stockPanel.add(stockLabel);
        stockPanel.add(stockSpacer);
        stockPanel.add(stockTextField);
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new BoxLayout(buttonPanel,
BoxLayout.LINE_AXIS));
        JButton goBackButton = new JButton("Go Back");
        JButton addToCartButton = new JButton("Add to Cart");
        if(itemCartIndex>=0)
        {addToCartButton.setText("Update Purchase Amount");}
        JButton viewDetailsButton = new JButton("View Details");
        buttonPanel.add(goBackButton);
        buttonPanel.add(addToCartButton);
        buttonPanel.add(viewDetailsButton);
        goBackButton.addActionListener((ActionEvent e)-
>{OpenShoppingScreen(currentCustomer);});
        addToCartButton.addActionListener((ActionEvent e)->
        {
            try
            {
                int stockCount = Integer.parseInt(stockTextField.getText());
                if(itemCartIndex>=0)
                {
                    if(stockCount>=0)
                    {
                        if(stockCount>item.GetStock())
                        {errorView.ShowError("This amount is over the current
stock of the item.");}
                        else
                        {

currentCustomer.customerCart.SetPurchaseAmount(itemCartIndex, stockCount);
                            OpenShoppingScreen(currentCustomer);
                        }
                    }else
                    {errorView.ShowError("When editing stock count for an
item in your cart please keep the stock equal to or over 0.");}
                }else{
```

```java
                    if(stockCount>0)
                    {
                        if(stockCount>item.GetStock())
                        {errorView.ShowError("This amount is over the current
stock of the item.");}
                        else
                        {

currentCustomer.customerCart.AddItem(item,stockCount);
                            OpenShoppingScreen(currentCustomer);
                        }
                    }else{errorView.ShowError("When adding a new item stock
must be greater than 0.");}
                }
            }catch(NumberFormatException ex)
            {errorView.ShowError("Invalid stock input.");}
        });
        viewDetailsButton.addActionListener((ActionEvent e)-
>{detailsView.ShowItem(item);});
        container.add(namePanel);
        container.add(descPanel);
        container.add(stockPanel);
        container.add(buttonPanel);
        mainView.pack();

    }

    /**
     * Opens the shopping cart screen for a specific customer.
     * This method constructs a graphical user interface displaying the items
     * currently added to the customer's shopping cart. It allows users to
view
     * the cart contents, modify the quantities of items, remove items, and
proceed
     * to complete the purchase transaction if all items are available and
valid.
     *
     * @param currentCustomer The customer whose shopping cart is being
displayed.
     */
    public void OpenShoppingCartScreen(Customer currentCustomer)
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container,BoxLayout.PAGE_AXIS));
        // Title Bar
        JPanel titleBar = new JPanel();
        JLabel titleLabel = new JLabel("My Shopping Cart");
        JButton logoutButton = new JButton("Log out");
        JButton goBackButton = new JButton("Go Back");
        JLabel balanceLabel = new JLabel("Balance:
"+String.valueOf(currentCustomer.GetBalance()));
        titleBar.add(titleLabel);
        titleBar.add(goBackButton);
        titleBar.add(logoutButton);
        titleBar.add(balanceLabel);
        logoutButton.addActionListener((ActionEvent e)->{
            currentUser = null;
            OpenLoginScreen();
        });
        goBackButton.addActionListener((ActionEvent e)->{
            OpenShoppingScreen(currentCustomer);
        });
        container.add(titleBar);
        // Cart Scroll Pane
        JScrollPane cartItemView = new
JScrollPane(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTAL_S
CROLLBAR_NEVER);
        container.add(cartItemView);
        JPanel cartPanel = new JPanel();
        ArrayList<JTextField> amountTextFields = new ArrayList<JTextField>();
        cartPanel.setLayout(new BoxLayout(cartPanel,BoxLayout.PAGE_AXIS));
        for(int i=0;i<currentCustomer.customerCart.GetSize();i++)
        {
            Item currentItem = currentCustomer.customerCart.GetItem(i);
            int myIndex = i;
            JPanel itemSuperPanel = new JPanel();
            itemSuperPanel.setLayout(new BoxLayout(itemSuperPanel,
BoxLayout.LINE_AXIS));
            JPanel itemLeftPanel = new JPanel();
            itemLeftPanel.setLayout(new BoxLayout(itemLeftPanel,
BoxLayout.PAGE_AXIS));
            JPanel itemCenterPanel = new JPanel();
            itemCenterPanel.setLayout(new BoxLayout(itemCenterPanel,
BoxLayout.PAGE_AXIS));
            JPanel itemRightPanel = new JPanel();
            itemRightPanel.setLayout(new BoxLayout(itemRightPanel,
BoxLayout.PAGE_AXIS));
            itemLeftPanel.add(new JLabel(currentItem.GetName()));
            itemLeftPanel.add(new JLabel("Seller:
"+currentItem.GetSeller().GetName()));
            itemCenterPanel.add(new JLabel("Price:
"+String.valueOf(currentItem.GetPrice())+"$"));
            JTextField purchaseAmountTextField = new
JTextField(String.valueOf(currentCustomer.customerCart.GetPurchaseAmount(myIn
dex)));
            purchaseAmountTextField.setColumns(3);
            purchaseAmountTextField.addMouseListener(new MouseListener(){
                @Override
                public void mouseClicked(MouseEvent e) {
                    purchaseAmountTextField.setBackground(Color.WHITE);
                }
                @Override
                public void mousePressed(MouseEvent e) {}
                @Override
                public void mouseReleased(MouseEvent e) {}
                @Override
```

```
    public void mouseEntered(MouseEvent e) {}
    @Override
    public void mouseExited(MouseEvent e) {}

});
itemCenterPanel.add(purchaseAmountTextField);
amountTextFields.add(purchaseAmountTextField);
JButton removeItemButton = new JButton("X");
removeItemButton.setBackground(Color.RED);
itemRightPanel.add(removeItemButton);
removeItemButton.addActionListener((ActionEvent e)->{
    currentCustomer.customerCart.RemoveItem(myIndex);
    OpenShoppingCartScreen(currentCustomer);
});
cartPanel.add(itemSuperPanel);
itemSuperPanel.add(itemLeftPanel);
JPanel leftSpacePanel = new JPanel();
leftSpacePanel.setSize(10, 0);
itemSuperPanel.add(leftSpacePanel);
itemSuperPanel.add(itemCenterPanel);
JPanel rightSpacePanel = new JPanel();
rightSpacePanel.setSize(10, 0);
itemSuperPanel.add(rightSpacePanel);
itemSuperPanel.add(itemRightPanel);
itemSuperPanel.addMouseListener(new MouseListener(){
    @Override
    public void mouseClicked(MouseEvent e)
    {detailsView.ShowItem(currentItem);}
    @Override
    public void mousePressed(MouseEvent e) {}
    @Override
    public void mouseReleased(MouseEvent e) {}
    @Override
    public void mouseEntered(MouseEvent e) {}
    @Override
    public void mouseExited(MouseEvent e) {}
});
if(myIndex%2==1){
    itemSuperPanel.setBackground(Color.LIGHT_GRAY);
    itemLeftPanel.setBackground(Color.LIGHT_GRAY);
    leftSpacePanel.setBackground(Color.LIGHT_GRAY);
    itemCenterPanel.setBackground(Color.LIGHT_GRAY);
    rightSpacePanel.setBackground(Color.LIGHT_GRAY);
    itemRightPanel.setBackground(Color.LIGHT_GRAY);
}
else{
    itemSuperPanel.setBackground(Color.WHITE);
    itemLeftPanel.setBackground(Color.WHITE);
    leftSpacePanel.setBackground(Color.WHITE);
    itemCenterPanel.setBackground(Color.WHITE);
    rightSpacePanel.setBackground(Color.WHITE);
    itemRightPanel.setBackground(Color.WHITE);
}
```

```
        }
        cartItemView.setViewportView(cartPanel);
        // Purchase Bar
        JPanel purchaseBar = new JPanel();
        purchaseBar.setLayout(new BoxLayout(purchaseBar,
BoxLayout.LINE_AXIS));
        JLabel totalPurchasePriceLabel = new JLabel("Total:
"+String.valueOf(currentCustomer.customerCart.GetTotal())+"$");
        JButton completeTransactionButton = new JButton("Complete
Transaction");
        purchaseBar.add(totalPurchasePriceLabel);
        purchaseBar.add(completeTransactionButton);
        completeTransactionButton.addActionListener((ActionEvent e)->{
            //errorView.ShowError("Transacting currently uncoded.");
            // Verify amounts are valid.
            if(currentCustomer.customerCart.GetSize()>0)
            {
                boolean flag = false;
                for(int i=0;i<amountTextFields.size();i++)
                {
                    try
                    {
                        String typedValue =
amountTextFields.get(i).getText();
                        int typedAmount = Integer.parseInt(typedValue);
                        if(
                                typedAmount<=0||

typedAmount>currentCustomer.customerCart.GetItem(i).GetStock()||

currentCustomer.customerCart.GetItem(i).GetAvailability()==false
                        )
                        {
                            flag = true;
                            amountTextFields.get(i).setBackground(Color.red);
                        }
                        else
                        {currentCustomer.customerCart.SetPurchaseAmount(i,
typedAmount);}
                    }
                    catch(NumberFormatException ex)
                    {
                        flag = true;
                        amountTextFields.get(i).setBackground(Color.red);
                    }
                }
                if(flag == true)
                {errorView.ShowError("Transaction failed, some of the items
are unavailabe or have an invalid amount typed. The other typed amounts have
been saved.");}
                else
                {
                    float cartTotal = 
currentCustomer.customerCart.GetTotal();
                    try
                    {
                        ArrayList<String[]> receipt = 
currentCustomer.CompleteTransaction();
                        OpenReceiptView(currentCustomer, receipt, cartTotal);
                    }
                    catch (Exception ex)
                    {errorView.ShowError("You lack the funds to complete this 
purchase.");}
                }
            }else{errorView.ShowError("There is nothing in your cart!");}
        });
        container.add(purchaseBar);
        mainView.pack();
    }

    /**
     * Opens a receipt view displaying the details of a completed 
transaction.
     * Constructs a graphical user interface to show the transaction receipt,
     * listing the purchased items, their prices, and quantities. Also 
includes
     * a "Done" button to return to the shopping screen.
     * @param currentCustomer The customer who completed the transaction.
     * @param receipt An ArrayList containing String arrays with item details 
for the receipt.
     * @param cartTotal The total cost of the items in the cart.
     */
    public void OpenReceiptView(Customer currentCustomer, ArrayList<String[]> 
receipt, float cartTotal)
    {
        ResetMainView();
        container.setLayout(new BorderLayout());
        Object[][] receipt2D = new Object[receipt.size()+2][3];
        receipt2D[0][0] = "Name";
        receipt2D[0][1] = "Price";
        receipt2D[0][2] = "Amount";
        for(int i=1;i<receipt.size()+1;i++)
        {
            System.arraycopy(receipt.get(i-1), 0, receipt2D[i], 0, 3);
        }
        receipt2D[receipt.size()+1][0] = "Total";
        receipt2D[receipt.size()+1][1] = String.valueOf(cartTotal);
        DefaultTableModel receiptTable = new 
DefaultTableModel(receipt2D,receipt2D[0]);
        JTable receiptDisplayTable = new JTable(receiptTable);
        DefaultTableCellRenderer receiptRenderer = new 
DefaultTableCellRenderer();

receiptRenderer.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
```

```
        receiptDisplayTable.setDefaultRenderer(Object.class,
receiptRenderer);
        JPanel receiptPanel = new JPanel(new BorderLayout());
        receiptPanel.add(receiptDisplayTable,BorderLayout.CENTER);
        JScrollPane receiptScrollPane = new JScrollPane(receiptPanel);

receiptScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLB
AR_NEVER);

receiptScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_A
S_NEEDED);
        JButton doneButton = new JButton("Done");
        doneButton.addActionListener((ActionEvent e)-
>{OpenShoppingScreen(currentCustomer);});
        container.add(new JLabel("Thank you for shopping!"),
BorderLayout.NORTH);
        container.add(receiptScrollPane,BorderLayout.CENTER);
        container.add(doneButton, BorderLayout.SOUTH);
        mainView.pack();
    }

    /**
     * Opens the inventory screen for a specific seller.
     * Constructs a graphical user interface to display the seller's
inventory,
     * allowing the seller to view their listed items, their availability,
prices, stock,
     * and remove items from sale. It also provides an option to add new
items.
     * @param currentSeller The seller whose inventory is being displayed.
     */
    public void OpenInventoryScreen(Seller currentSeller)
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container, BoxLayout.PAGE_AXIS));
        // Title Bar
        JPanel titleBar = new JPanel();
        JLabel titleLabel = new JLabel("Inventory");
        JButton logoutButton = new JButton("Log out");
        JButton settingsButton = new JButton("User Settings");
        JLabel balanceLabel = new JLabel("Balance:
"+String.valueOf(currentSeller.GetBalance()));
        titleBar.add(titleLabel);
        titleBar.add(logoutButton);
        titleBar.add(settingsButton);
        titleBar.add(balanceLabel);
        logoutButton.addActionListener((ActionEvent e)->{
            currentUser = null;
            OpenLoginScreen();
        });
        settingsButton.addActionListener((ActionEvent e)->{
            OpenSettingsScreen();
        });
        container.add(titleBar);
        // Inventory List
        JPanel inventoryPanel = new JPanel();
        inventoryPanel.setLayout(new
BoxLayout(inventoryPanel,BoxLayout.PAGE_AXIS));
        // Looping through the seller's inventory items
        for(int i=0;i<currentSeller.GetInventorySize();i++)
        {
            Item currentItem = currentSeller.GetItemInInventory(i);
            int myIndex = i;
            JPanel itemSuperPanel = new JPanel();
            itemSuperPanel.setLayout(new BoxLayout(itemSuperPanel,
BoxLayout.LINE_AXIS));
            JPanel itemLeftPanel = new JPanel();
            itemLeftPanel.setLayout(new BoxLayout(itemLeftPanel,
BoxLayout.PAGE_AXIS));
            JPanel itemRightPanel = new JPanel();
            itemRightPanel.setLayout(new BoxLayout(itemRightPanel,
BoxLayout.PAGE_AXIS));
            JPanel itemRemovePanel = new JPanel();
            itemRemovePanel.setLayout(new BoxLayout(itemRemovePanel,
BoxLayout.PAGE_AXIS));
            itemLeftPanel.add(new JLabel(currentItem.GetName()));
            itemLeftPanel.add(new JLabel("On Sale:
"+String.valueOf(currentItem.GetAvailability())));
            itemRightPanel.add(new JLabel("Price:
"+String.valueOf(currentItem.GetPrice())+"$"));
            itemRightPanel.add(new JLabel("Stock:
"+String.valueOf(currentItem.GetStock())));
            JButton itemRemoveButton = new JButton("X");
            itemRemoveButton.setBackground(Color.RED);
            itemRemoveButton.addActionListener((ActionEvent e)->{
                currentSeller.RemoveFromSale(myIndex);
                OpenInventoryScreen(currentSeller);
            });
            itemRemovePanel.add(itemRemoveButton);
            inventoryPanel.add(itemSuperPanel);
            itemSuperPanel.add(itemLeftPanel);
            JPanel spacePanel = new JPanel();
            spacePanel.setSize(10, 0);
            itemSuperPanel.add(spacePanel);
            itemSuperPanel.add(itemRightPanel);
            JPanel removeSpacePanel = new JPanel();
            removeSpacePanel.setSize(10, 0);
            itemSuperPanel.add(removeSpacePanel);
            itemSuperPanel.add(itemRemovePanel);
            itemSuperPanel.addMouseListener(new MouseListener(){
                @Override
                public void mouseClicked(MouseEvent e)
                {OpenItemEditScreen(currentSeller,myIndex);}
                @Override
                public void mousePressed(MouseEvent e) {}
                @Override
                public void mouseReleased(MouseEvent e) {}
                @Override
                public void mouseEntered(MouseEvent e) {}
                @Override
                public void mouseExited(MouseEvent e) {}
            });
            if(myIndex%2==1){
                itemSuperPanel.setBackground(Color.LIGHT_GRAY);
                itemLeftPanel.setBackground(Color.LIGHT_GRAY);
                spacePanel.setBackground(Color.LIGHT_GRAY);
                itemRightPanel.setBackground(Color.LIGHT_GRAY);
                removeSpacePanel.setBackground(Color.LIGHT_GRAY);
                itemRemovePanel.setBackground(Color.LIGHT_GRAY);
            }
            else{
                itemSuperPanel.setBackground(Color.WHITE);
                itemLeftPanel.setBackground(Color.WHITE);
                spacePanel.setBackground(Color.WHITE);
                itemRightPanel.setBackground(Color.WHITE);
                removeSpacePanel.setBackground(Color.WHITE);
                itemRemovePanel.setBackground(Color.WHITE);
            }
        }
        JButton newItemButton = new JButton("Add New Item");
        inventoryPanel.add(newItemButton);
        newItemButton.addActionListener((ActionEvent e)->
        {OpenItemEditScreen(currentSeller,-1);});
        newItemButton.setAlignmentX(JButton.CENTER_ALIGNMENT);
        JScrollPane inventoryView = new JScrollPane(inventoryPanel);

inventoryView.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_N
EVER);

inventoryView.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NE
EDED);
        container.add(inventoryView);
        mainView.pack();
    }

    /**
     * Opens the screen for editing an item's details.
     * If editing an existing item, it retrieves the item's details and sets
up the screen.
     * If adding a new item, initializes an empty item for editing.
     * @param currentSeller The seller whose item is being edited or for whom
a new item is being added.
     * @param itemIndex The index of the item being edited in the seller's
inventory.
     *                 Use -1 to initialize a new item for addition.
     */
    public void OpenItemEditScreen(Seller currentSeller, int itemIndex)
    {
        ResetMainView();
        container.setLayout(new BoxLayout(container,BoxLayout.PAGE_AXIS));
        // Get Values
        Item item;
        if(itemIndex>=0)
        {item = currentSeller.GetItemInInventory(itemIndex);}
        else
        {item = new Item("","",0,new ArrayList<String>(),new
ArrayList<String>(),currentSeller);}
        String tempName = item.GetName();
        String tempDescription = item.GetDescription();
        float tempPrice = item.GetPrice();
        int tempStock = item.GetStock();
        boolean tempAvailability = item.GetAvailability();
        ArrayList<String> tempSpecName = new ArrayList<String>();
        ArrayList<String> tempSpec = new ArrayList<String>();
        for(int i=0;i<item.GetSpecSize();i++)
        {
            String[] specInfo = item.GetSpec(i);
            tempSpecName.add(specInfo[0]);
            tempSpec.add(specInfo[1]);
        }
        // Edit Item View
        MouseListener textFieldListener = new MouseListener(){
            @Override
            public void mouseClicked(MouseEvent e)
            {e.getComponent().setBackground(Color.WHITE);}
            @Override
            public void mousePressed(MouseEvent e) {}
            @Override
            public void mouseReleased(MouseEvent e) {}
            @Override
            public void mouseEntered(MouseEvent e) {}
            @Override
            public void mouseExited(MouseEvent e) {}
        };
        // Name
        JPanel itemNamePanel = new JPanel();
        itemNamePanel.setLayout(new BoxLayout(itemNamePanel,
BoxLayout.LINE_AXIS));
        JLabel itemNameLabel = new JLabel("Name:");
        JTextField itemNameTextField = new JTextField(tempName);
        itemNameTextField.setColumns(20);
        itemNamePanel.add(itemNameLabel);
        itemNamePanel.add(itemNameTextField);
        itemNameTextField.addMouseListener(textFieldListener);
        container.add(itemNamePanel);
        // Description
        JPanel itemDescriptionPanel = new JPanel();
        itemDescriptionPanel.setLayout(new BoxLayout(itemDescriptionPanel,
BoxLayout.LINE_AXIS));
        JLabel itemDescriptionLabel = new JLabel("Description:");
        JTextField itemDescriptionTextField = new
JTextField(tempDescription);
```

```
        itemDescriptionTextField.setColumns(20);
        itemDescriptionPanel.add(itemDescriptionLabel);
        itemDescriptionPanel.add(itemDescriptionTextField);
        itemDescriptionTextField.addMouseListener(textFieldListener);
        container.add(itemDescriptionPanel);
        // Price
        JPanel itemPricePanel = new JPanel();
        itemPricePanel.setLayout(new BoxLayout(itemPricePanel,
BoxLayout.LINE_AXIS));
        JLabel itemPriceLabel = new JLabel("Price:");
        JTextField itemPriceTextField = new
JTextField(String.valueOf(tempPrice));
        itemPriceTextField.setColumns(3);
        itemPricePanel.add(itemPriceLabel);
        itemPricePanel.add(itemPriceTextField);
        itemPriceTextField.addMouseListener(textFieldListener);
        container.add(itemPricePanel);
        // Stock
        JPanel itemStockPanel = new JPanel();
        itemStockPanel.setLayout(new BoxLayout(itemStockPanel,
BoxLayout.LINE_AXIS));
        JLabel itemStockLabel = new JLabel("Amount in stock:");
        JTextField itemStockTextField = new
JTextField(String.valueOf(tempStock));
        itemStockTextField.setColumns(3);
        itemStockPanel.add(itemStockLabel);
        itemStockPanel.add(itemStockTextField);
        itemStockTextField.addMouseListener(textFieldListener);
        container.add(itemStockPanel);
        // Available
        JPanel itemAvailabilityPanel = new JPanel();
        itemAvailabilityPanel.setLayout(new BoxLayout(itemAvailabilityPanel,
BoxLayout.LINE_AXIS));
        JCheckBox itemAvailableCheckBox = new JCheckBox("Item is on sale");
        itemAvailableCheckBox.setSelected(tempAvailability);
        itemAvailabilityPanel.add(itemAvailableCheckBox);
        container.add(itemAvailabilityPanel);
        // Specs Scrollpane
        JPanel itemSpecsPanel = new JPanel();
        ArrayList<JTextField> itemSpecNameTFList = new
ArrayList<JTextField>();
        ArrayList<JTextField> itemSpecTFList = new ArrayList<JTextField>();
        itemSpecsPanel.setLayout(new BoxLayout(itemSpecsPanel,
BoxLayout.PAGE_AXIS));
        for(int i=0;i<tempSpecName.size();i++)
        {
            int myIndex = i;
            JPanel specPanel = new JPanel();
            specPanel.setLayout(new BoxLayout(specPanel,
BoxLayout.LINE_AXIS));
            JTextField specNameTF = new
JTextField(tempSpecName.get(myIndex));
            JTextField specValueTF = new JTextField(tempSpec.get(myIndex));
            JButton deleteSpec = new JButton("X");
            deleteSpec.setBackground(Color.red);
            specPanel.add(specNameTF);
            specPanel.add(specValueTF);
            specPanel.add(deleteSpec);
            specNameTF.addMouseListener(textFieldListener);
            specValueTF.addMouseListener(textFieldListener);
            itemSpecNameTFList.add(specNameTF);
            itemSpecTFList.add(specValueTF);
            deleteSpec.addActionListener((ActionEvent e)->{
                itemSpecsPanel.remove(specPanel);
                int myCurrentIndex = itemSpecNameTFList.indexOf(specNameTF);
                itemSpecNameTFList.remove(myCurrentIndex);
                itemSpecTFList.remove(myCurrentIndex);
                tempSpec.remove(myCurrentIndex);
                tempSpecName.remove(myCurrentIndex);
                itemSpecsPanel.revalidate();
                itemSpecsPanel.repaint();
                mainView.pack();
            });
            itemSpecsPanel.add(specPanel);
        }
        JPanel newSpec = new JPanel();
        newSpec.setLayout(new FlowLayout());
        JButton newSpecButton = new JButton("Add new specification");
        newSpec.add(newSpecButton);
        newSpecButton.addActionListener((ActionEvent e)->{
            itemSpecsPanel.remove(newSpec);
            int myIndex = tempSpecName.size();
            tempSpecName.add("");
            tempSpec.add("");
            JPanel specPanel = new JPanel();
            specPanel.setLayout(new BoxLayout(specPanel,
BoxLayout.LINE_AXIS));
            JTextField specNameTF = new
JTextField(tempSpecName.get(myIndex));
            JTextField specValueTF = new JTextField(tempSpec.get(myIndex));
            JButton deleteSpec = new JButton("X");
            deleteSpec.setBackground(Color.red);
            specPanel.add(specNameTF);
            specPanel.add(specValueTF);
            specPanel.add(deleteSpec);
            specNameTF.addMouseListener(textFieldListener);
            specValueTF.addMouseListener(textFieldListener);
            itemSpecNameTFList.add(specNameTF);
            itemSpecTFList.add(specValueTF);
            deleteSpec.addActionListener((ActionEvent e2)->{
                itemSpecsPanel.remove(specPanel);
                int myCurrentIndex = itemSpecNameTFList.indexOf(specNameTF);
                itemSpecNameTFList.remove(myCurrentIndex);
                itemSpecTFList.remove(myCurrentIndex);
                tempSpec.remove(myCurrentIndex);
                tempSpecName.remove(myCurrentIndex);
                itemSpecsPanel.revalidate();
                itemSpecsPanel.repaint();
                mainView.pack();
            });
            itemSpecsPanel.add(specPanel);
            itemSpecsPanel.add(newSpec);
            itemSpecsPanel.revalidate();
            itemSpecsPanel.repaint();
            mainView.pack();
        });
        itemSpecsPanel.add(newSpec);
        JScrollPane itemSpecScrollPane = new JScrollPane(itemSpecsPanel);

itemSpecScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLL
BAR_NEVER);

itemSpecScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_
AS_NEEDED);
        container.add(itemSpecScrollPane);
        // Return Button
        JPanel actionButtonPanel = new JPanel();
        actionButtonPanel.setLayout(new
BoxLayout(actionButtonPanel,BoxLayout.LINE_AXIS));
        JButton returnButton = new JButton("Go Back");
        actionButtonPanel.add(returnButton);
        returnButton.addActionListener((ActionEvent e)->{
            OpenInventoryScreen(currentSeller);
        });
        // Preview Button
        if(itemIndex>=0)
        {
            JButton previewButton = new JButton("View as Customer");
            actionButtonPanel.add(previewButton);
            previewButton.addActionListener((ActionEvent e)->
            {detailsView.ShowItem(item);});
        }

        /**
        * Constructs a JButton labeled "Save" or "Add" based on the item
index.
        * Adds an ActionListener to the button to handle saving or adding
item details.
        * Highlights text fields with red if they are empty or contain
invalid inputs.
        * Displays appropriate error messages for invalid inputs or actions.
        * Updates item details and performs actions based on the conditions.
        */
        // Save Button
        JButton saveButton = new JButton("Save");
        if(itemIndex==-1)
        {saveButton.setText("Add");}
        actionButtonPanel.add(saveButton);
        saveButton.addActionListener((ActionEvent e)->{
            // Collate Text Fields
            ArrayList<JTextField> textFieldChecks = new
ArrayList<JTextField>();
            textFieldChecks.add(itemNameTextField);
            textFieldChecks.add(itemDescriptionTextField);
            textFieldChecks.add(itemPriceTextField);
            textFieldChecks.add(itemStockTextField);
            for(int i=0;i<itemSpecNameTFList.size();i++)
            {textFieldChecks.add(itemSpecNameTFList.get(i));}
            for(int i=0;i<itemSpecTFList.size();i++)
            {textFieldChecks.add(itemSpecTFList.get(i));}
            // Check for Problems
            boolean flag = false;
            for(int i=0;i<textFieldChecks.size();i++)
            {
                if(textFieldChecks.get(i).getText().equals(""))
                {
                    flag = true;
                    textFieldChecks.get(i).setBackground(Color.red);
                }
            }
            if(flag)
            {
                errorView.ShowError("One or more of the textfields
(highlighted in red) are empty.");
                return;
            }
            float inputPriceValue;
            try
            {inputPriceValue =
Float.parseFloat(itemPriceTextField.getText());}
            catch(NumberFormatException ex)
            {
                errorView.ShowError("Invalid input price.");
                itemPriceTextField.setBackground(Color.red);
                return;
            }
            if (inputPriceValue<0.01)
            {
                errorView.ShowError("Price is too small.");
                itemPriceTextField.setBackground(Color.red);
                return;
            }
            int inputStockValue;
            try
            {inputStockValue =
Integer.parseInt(itemStockTextField.getText());}
            catch(NumberFormatException ex)
            {
                errorView.ShowError("Invalid stock input.");
                itemStockTextField.setBackground(Color.red);
                return;
            }
            if(inputStockValue<0)
            {
                errorView.ShowError("Stock can't be less than 0.");
                itemStockTextField.setBackground(Color.red);
                return;
            }
            if(inputStockValue==0&&itemAvailableCheckBox.isSelected())
            {
                errorView.ShowError("When stock is 0 the item can't be set to
available.");
                return;
            }
            // Fill Spec Sheet
            for(int i=0;i<tempSpecName.size();i++)
            {tempSpecName.set(i, itemSpecNameTFList.get(i).getText());}
            for(int i=0;i<tempSpec.size();i++)
            {tempSpec.set(i, itemSpecTFList.get(i).getText());}
            // Saving
            item.SetName(itemNameTextField.getText());
            item.SetDescription(itemDescriptionTextField.getText());
            item.SetPrice(inputPriceValue);
            try{item.SetStock(inputStockValue);}catch(Exception ex){}

try{item.SetAvailability(itemAvailableCheckBox.isSelected());}catch(Exception
ex){}
            item.SetSpecNames(tempSpecName);
            item.SetSpecs(tempSpec);
            if(itemIndex==-1)
            {
                currentSeller.SellItem(item);
                detailsView.ShowItem(item);
                OpenInventoryScreen(currentSeller);
            }else
            {errorView.showStatusSuccess();}
        });
        container.add(actionButtonPanel);
        mainView.pack();
    }

   /**
    * Retrieves a seller object based on the provided username.
    * @param username The username of the seller to search for.
    * @return The Seller object associated with the provided username.
    * @throws NoSuchElementException If no seller is found with the given
username.
    */
    public Seller GetSellerFromUsername(String username) throws
NoSuchElementException {
        Seller foundSeller = null;
        for (int i = 0; i < sellersList.size(); i++) {
            Seller currentSeller = sellersList.get(i);
            if (currentSeller.GetUsername().equals(username)) {
                foundSeller = currentSeller;
                break;
            }
        }
        if (foundSeller == null)
            throw new NoSuchElementException();
        return foundSeller;
    }

    /**
     * Retrieves a customer object based on the provided username.
     * @param username The username of the customer to search for.
     * @return The Customer object associated with the provided username.
     * @throws NoSuchElementException If no customer is found with the given
username.
     */
    public Customer GetCustomerFromUsername(String username) throws
NoSuchElementException {
        Customer foundCustomer = null;
        for (int i = 0; i < customersList.size(); i++) {
            Customer currentCustomer = customersList.get(i);
            if (currentCustomer.GetUsername().equals(username)) {
                foundCustomer = currentCustomer;
                break;
            }
        }
        if (foundCustomer == null)
            throw new NoSuchElementException();
        return foundCustomer;
    }

    /**
     * Checks if a username is taken by either a seller or a customer.
     * @param testUsername The username to be checked for availability.
     * @return True if the username is taken by either a seller or a
customer, otherwise false.
     */
    public boolean UsernameTaken(String testUsername) {
        try {
            GetSellerFromUsername(testUsername);
            return true;
        } catch (NoSuchElementException e1) {
            try {
                GetCustomerFromUsername(testUsername);
                return true;
            } catch (NoSuchElementException e2) {
                return false;
            }
        }
    }

    /**
     * Main entry point of the application. Initiates the starting of the
system.
```

```
     * @param Args The command line arguments provided to the program.
     */
    public static void main(String[] Args)
    {systemSingleton.StartApp();}

    /**
     * Retrieves the instance of the ShoppingSystem (singleton pattern).
     * @return The singleton instance of the ShoppingSystem.
     */
    public static ShoppingSystem GetInstance()
    {return systemSingleton;}

    private static final ShoppingSystem systemSingleton = new
ShoppingSystem();
    private User currentUser;
    private ArrayList<Seller> sellersList;
    private ArrayList<Customer> customersList;
    private final JFrame mainView = new JFrame();
    private final JPanel container = new JPanel();
    private final ErrorView errorView = ErrorView.GetInstance();
    private final ItemDetailsView detailsView =
ItemDetailsView.GetInstance();
}
```

JAVA CODE : MODELS

Customer.java

```
package cop4331.models;

import cop4331.gui.ErrorView;
import java.util.ArrayList;

/**
 * Represents a customer user within the system, extending the User class.
 * Manages transactions and the customer's shopping cart.
 * Provides functionality to complete transactions and handle cart
operations.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class Customer extends User{
    /**
     * Constructor for creating a Customer object with username, password,
and name.
     * Initializes the user with a role of "Customer".
     * @param username The username for the customer.
     * @param password The password for the customer.
     * @param name The name of the customer.
     */
    public Customer(String username, String password, String name) {
        super(username, password, name, "Customer");
    }

    /**
     * Completes a transaction, updating stock, funds, and generating a
receipt.
     * Removes funds from the customer's account based on the cart's total.
     * Processes items in the cart, updates stock, and generates a receipt.
     * @return An ArrayList containing String arrays representing transaction
information.
     * @throws Exception If an error occurs during the transaction process.
     */
    public ArrayList<String[]> CompleteTransaction() throws Exception
    {
        RemoveFunds(customerCart.GetTotal());
        ArrayList<String[]> Receipt = new ArrayList<String[]>();
        // Process items in the cart
        for(int i=0;i<customerCart.GetSize();i++)
        {
            Item indexedItem = customerCart.GetItem(i);
            try{
                float itemPurchasePriceTotal =
customerCart.GetPurchaseAmount(i)*indexedItem.GetPrice();
                indexedItem.ChangeStock(customerCart.GetPurchaseAmount(i)*-
1);
                indexedItem.GetSeller().AddFunds(itemPurchasePriceTotal);
                String[] InfoArray =
                {
                    indexedItem.GetName(),
                    String.valueOf(indexedItem.GetPrice()),
                    String.valueOf(customerCart.GetPurchaseAmount(i))
                };
                Receipt.add(InfoArray);
            }catch(Exception e)
            {ErrorView.GetInstance().ShowError("An error occured with
purchasing one or more item(s).");}
        }
        customerCart.ClearCart();
        return Receipt;
    }

    /** The customer's shopping cart containing items for purchase. */
    public final ShoppingCart customerCart = new ShoppingCart();
}
```

Item.java

```java
package cop4331.models;

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

/**
 * Represents an item available for sale within the system.
 * Implements Serializable to enable serialization of Item objects.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class Item implements Serializable {
    /**
     * Constructor for creating an Item object with specific details.
     * Initializes the item with provided attributes.
     * @param name The name of the item.
     * @param description The description of the item.
     * @param price The price of the item.
     * @param specsName The list of specification names for the item.
     * @param specs The list of specifications for the item.
     * @param soldBy The seller who sells this item.
     */
    public Item(String name, String description, float price,
ArrayList<String> specsName, ArrayList<String> specs, Seller soldBy)
    {
        // Initialization of item attributes
        this.name = name;
        this.description = description;
        this.price = price;
        this.specsName = specsName;
        this.specs = specs;
        this.soldBy = soldBy;
        amountInStock = 0;
        available = false;
    }

    // Getter methods for item attributes
    /**
     * Retrieves the name of the item.
     * @return The name of the item as a String.
     */
    public String GetName() {
        return name;
    }
    /**
     * Retrieves the description of the item.
     * @return The description of the item as a String.
     */
    public String GetDescription() {
        return description;
    }

    /**
     * Retrieves the price of the item.
     * @return The price of the item as a float value.
     */
    public float GetPrice() {
        return price;
    }

    /**
     * Retrieves the amount of the item in stock.
     * @return The amount of the item in stock as an integer.
     */
    public int GetStock() {
        return amountInStock;
    }

    /**
     * Retrieves the size of the specifications list.
     * @return The size of the specifications list as an integer.
     */
    public int GetSpecSize() {
        return specs.size();
    }

    /**
     * Retrieves the availability status of the item.
     * @return The availability status of the item as a boolean.
     */
    public boolean GetAvailability() {
        return available;
    }

    /**
     * Retrieves the seller of the item.
     * @return The Seller object representing the seller of the item.
     */
    public Seller GetSeller() {
        return soldBy;
    }


   /**
    * Retrieves the specifications for an item at a given index.
    * @param index The index of the specifications to retrieve.
    * @return An array containing the specification name and details at the
given index.
    */
   public String[] GetSpec(int index) {
       String specName = specsName.get(index);
       String spec = specs.get(index);
       String[] specification = {specName, spec};
       return specification;
   }

    /**
     * Changes the stock amount of the item by a specified amount.
     * @param amount The amount to be added to the current stock.
     * @throws Exception If an invalid amount is provided (e.g., negative
amount).
     */
    public void ChangeStock(int amount) throws Exception {
        SetStock(amountInStock + amount);
    }

    /**
     * Sets the stock amount of the item to a specified value.
     * @param amount The new stock amount to be set.
     * @throws Exception If an invalid amount (negative) is attempted to set.
     */
    public void SetStock(int amount) throws Exception {
        if (amount < 0)
            throw new Exception();
        amountInStock = amount;
        if (amount == 0)
            available = false;
    }

    /**
     * Sets the availability status of the item.
     * @param available The new availability status to be set.
     * @throws Exception If the item's stock is 0 and an attempt is made to
set it as available.
     */
    public void SetAvailability(boolean available) throws Exception {
        if (amountInStock == 0 && available == true)
            throw new Exception();
        this.available = available;
    }

    /**
     * Sets the name of the item.
     * @param name The new name for the item.
     */
    public void SetName(String name)
    {this.name=name;}

    // Other setter methods for description, price, stock, etc.
```

```java
    /**
     * Sets the description of the item.
     * @param description The new description to be set.
     */
    public void SetDescription(String description) {
        this.description = description;
    }

    /**
     * Sets the price of the item.
     * @param price The new price to be set.
     */
    public void SetPrice(float price) {
        this.price = price;
    }

    /**
     * Sets the specifications of the item.
     * @param specs The new specifications to be set (as an ArrayList of
strings).
     */
    public void SetSpecs(ArrayList<String> specs) {
        this.specs = specs;
    }

    /**
     * Sets the specification names of the item.
     * @param specsName The new specification names to be set (as an
ArrayList of strings).
     */
    public void SetSpecNames(ArrayList<String> specsName) {
        this.specsName = specsName;
    }

    /**
     * Sets the seller of the item.
     * @param soldBy The Seller object representing the new seller to be set.
     */
    public void SetSeller(Seller soldBy) {
        this.soldBy = soldBy;
    }

    /**
     * Checks equality between two Item objects based on their attributes.
     * @param other The object to compare for equality.
     * @return True if the objects are equal, false otherwise.
     */
    @Override
    public boolean equals(Object other)
    {
        // Check for equality based on attributes
        if(other == null||getClass() != other.getClass())
            return false;
        Item otherItem = (Item) other;
        return(this.hashCode() == otherItem.hashCode());
    }

    /**
     * Generates a hash code for the Item object.
     * @return The generated hash code for the Item object.
     */
    @Override
    public int hashCode() {
        // Generate a hash code based on attributes
        int hash = 5;
        hash = 97 * hash + Objects.hashCode(this.name);
        hash = 97 * hash + Objects.hashCode(this.description);
        hash = 97 * hash + Float.floatToIntBits(this.price);
        hash = 97 * hash + Objects.hashCode(this.specs);
        hash = 97 * hash + Objects.hashCode(this.specsName);
        return hash;
    }

    /** The name of the item. */
    private String name;
    /** The description of the item. */
    private String description;
    /** The price of the item. */
    private float price;
    /** The amount of the item currently in stock. */
    private int amountInStock;
    /** The availability status of the item.. */
    private boolean available;
    /** The seller of the item. */
    private transient Seller soldBy;
    /** A list of specs for the item. */
    private ArrayList<String> specs;
    /** The associated names for the list of specs. */
    private ArrayList<String> specsName;
}
```

Seller.java

```java
package cop4331.models;

import java.util.ArrayList;

/**
 * Represents a seller within the system, extending the User class.
 * Manages inventory and selling of items.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class Seller extends User {
    /**
     * Constructor for creating a Seller object with username, password, and
name.
     * Initializes the user with a role of "Seller" and an empty inventory.
     * @param username The username for the seller.
     * @param password The password for the seller.
     * @param name The name of the seller.
     */
    public Seller(String username, String password, String name) {


        super(username, password, name, "Seller");
        inventory = new ArrayList<Item>();
    }

    /**
     * Adds an item to the seller's inventory.
     * @param product The item to be added to the inventory.
     */
    public void SellItem(Item product)
    {
        inventory.add(product);
    }

    /**
     * Adds an item with a specified stock to the seller's inventory.
     * @param product The item to be added to the inventory.
     * @param stock The stock quantity for the item.
     */
    public void SellItem(Item product, int stock)
    {
        try {product.SetStock(stock);} catch (Exception ex) {}
        try {product.SetAvailability(true);}catch (Exception ex){}
        SellItem(product);
    }
    /**
     * Updates an item in the seller's inventory at a specific index.
     * @param updatedProduct The updated item to replace the existing one.
     * @param index The index of the item to be updated.
     */
    public void UpdateItem(Item updatedProduct, int index)
    {inventory.set(index, updatedProduct);}

    // Other methods for inventory management (size, retrieval, removal,
etc.)
    /**
     * Retrieves the size of the inventory.
     * @return The size of the inventory as an integer.
     */
    public int GetInventorySize() {
        return inventory.size();
    }

    /**
     * Retrieves the item at the specified index in the inventory.
     * @param index The index of the item to retrieve from the inventory.
     * @return The Item object at the specified index in the inventory.
     */
    public Item GetItemInInventory(int index) {
        return inventory.get(index);
    }

    /**
     * Removes an item from the sale at the specified index.
     * @param index The index of the item to remove from sale.
     */
    public void RemoveFromSale(int index) {
        try {
            inventory.get(index).SetAvailability(false);
        } catch (Exception e) {
            // Exception handling if an error occurs when setting
availability
        }
        inventory.remove(index);
    }

    /**
     * Changes the stock amount of the item at the specified index in the
inventory.
     * @param index The index of the item in the inventory.
     * @param amount The new stock amount to be set.
     * @throws Exception If an invalid stock amount is attempted to set
(e.g., negative amount).
     */
    public void ChangeStock(int index, int amount) throws Exception {
        GetItemInInventory(index).SetStock(amount);
    }
    /**
     * Changes the availability of the item at the specified index in the
inventory.
     * @param index The index of the item in the inventory.
     * @param available The new availability status to be set.
     * @throws Exception If an invalid availability status is attempted to
set based on stock quantity.
     */
    public void ChangeAvailability(int index, boolean available) throws
Exception {
        GetItemInInventory(index).SetAvailability(available);
    }

    // Private attribute representing the inventory
    /** The inventory of the seller. */
    private final ArrayList<Item> inventory;
}
```

ShoppingCart.java

```java
package cop4331.models;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * Represents a shopping cart for items to be purchased.
 * Implements Serializable to enable serialization of ShoppingCart objects.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class ShoppingCart implements Serializable{
    /**
     * Default constructor for the shopping cart.
     */
    public ShoppingCart(){}

    /**
     * Retrieves the item at a specific index in the shopping cart.
     * @param index The index of the item to retrieve.
     * @return The item at the specified index in the shopping cart.
     */
    public Item GetItem(int index)
    {return items.get(index);}

    // Other getter and utility methods for shopping cart management
    /**
     * Retrieves the purchase amount of an item at a specified index in the
cart.
     * @param index The index of the item in the cart.
     * @return The purchase amount of the item at the specified index.
     */
    public int GetPurchaseAmount(int index) {
        return amountPurchasing.get(index);
    }

    /**
     * Checks if the cart contains a specific item and returns its index.
     * @param item The item to check for in the cart.
     * @return The index of the item in the cart; returns -1 if the item is
not present.
     */
    public int ContainsItem(Item item) {
        return items.indexOf(item);
    }

    /**
     * Sets the purchase amount for an item at a specified index in the cart.
     * @param index The index of the item in the cart.
     * @param amount The new purchase amount to be set.
     * If the amount is 0 or less, the item is removed from the cart.
     */
    public void SetPurchaseAmount(int index, int amount) {
        if (amount > 0)
            amountPurchasing.set(index, amount);
        else
            RemoveItem(index);
    }

    /**
     * Removes an item from the cart at the specified index.
     * @param index The index of the item to be removed from the cart.
     */
    public void RemoveItem(int index) {
        items.remove(index);
        amountPurchasing.remove(index);
    }

    /**
     * Clears the entire cart by removing all items and purchase amounts.
     */
    public void ClearCart() {
        items.clear();
        amountPurchasing.clear();
    }

    /**
     * Adds an item to the cart with a default purchase amount of 1.
     * @param newItem The item to be added to the cart.
     */
    public void AddItem(Item newItem) {
        items.add(newItem);
        amountPurchasing.add(1);
    }

    /**
     * Adds an item to the shopping cart with a specified purchase amount.
     * @param newItem The item to add to the shopping cart.
     * @param amount The purchase amount for the item.
     */
    public void AddItem(Item newItem, int amount)
    {
        items.add(newItem);
        amountPurchasing.add(amount);
    }

    /**
     * Gets the number of items in a shopping cart.
     * @return The size of the shopping cart.
     */
```

```
    public int GetSize()
    {return items.size();}

    // Other methods for managing items in the shopping cart
    /**
     * Retrieves the total price of items in the shopping cart.
     * @return The total price of items in the shopping cart.
     */
    public float GetTotal()
    // Calculates the total price of items in the cart
    {
        float total = 0;
        int cartSize = GetSize();
        for(int i=0; i<cartSize; i++)
        {total += (GetItem(i).GetPrice()*amountPurchasing.get(i));}
        return total;
    }

    /** The array representing the items in the shopping cart. */
    private final ArrayList<Item> items = new ArrayList<>();
    /** The array representing the amount of items being purchased for each
item in the shopping cart. */
    private final ArrayList<Integer> amountPurchasing = new ArrayList<>();
}
```

User.java

```
package cop4331.models;

import java.io.Serializable;

/**test
 * Represents a user within the system.
 * Implements Serializable to enable serialization of User objects.
 * @author S Hassan Shaikh
 * @author Austin Vasquez
 * @author Divyesh Mangapuram
 * @author Jorge Martinez
 */
public class User implements Serializable {
    /**
     * Constructor for creating a User object with username, password, name,
and user type.
     * @param username The username for the user.
     * @param password The password for the user.
     * @param name The name of the user.
     * @param userType The type of user (e.g., "Seller", "Customer").
     */
    public User(String username, String password, String name, String
userType)
    {
        // Initialization of user attributes
        this.username = username;
        this.password = password;
        this.name = name;
        this.userType = userType;
    }

    /**
     * Changes the username of the user.
     * @param newUsername The new username to set for the user.
     */
    public void ChangeUsername(String newUsername)
    {username = newUsername;}

    /**
     * Changes the user's password if the old password matches.
     * @param oldPassword The old password to be verified.
     * @param newPassword The new password to be set if the old password
matches.
     * @throws Exception If the old password doesn't match, preventing the
password change.
     */
    public void ChangePassword(String oldPassword, String newPassword) throws
Exception {
        if (CheckPassword(oldPassword)) {
            password = newPassword;
        } else {
            throw new Exception();
        }
    }

    /**
     * Adds funds to the user's balance.
     * @param amount The amount of funds to be added.
     * @throws Exception If an invalid amount (negative) is attempted to add.
     */
    public void AddFunds(float amount) throws Exception {
        if (amount >= 0) {
            balance += amount;
        } else {
            throw new Exception();
        }
    }

    /**
     * Removes funds from the user's balance if sufficient funds are
available.
     * @param amount The amount of funds to be removed.
     * @throws Exception If an invalid amount (negative or exceeding balance)
is attempted to remove.
     */
    public void RemoveFunds(float amount) throws Exception {
        if (amount >= 0 && amount <= balance) {
            balance -= amount;
        } else {
            throw new Exception();
        }
    }

    /**
     * Changes the name associated with an object.
     * @param newName The new name to be assigned to the object.
     */
    public void ChangeName(String newName)
    {name = newName;}

    /**
     * Retrieves the name of the user.
     *
     * @return The name of the user.
     */
    public String GetName()
    {return name;}

    // Other getter methods for username, user type, balance, etc.
    /**
     * Retrieves the username of the user.
     * @return The username of the user as a String.
     */
    public String GetUsername() {
        return username;
    }

    /**
     * Retrieves the type of user.
     * @return The type of user as a String.
     */
    public String GetUserType() {
        return userType;
    }

    /**
     * Checks if the provided input matches the user's password.
     *
     * @param input The input to check against the user's password.
     * @return True if the input matches the password, false otherwise.
     */
    public boolean CheckPassword(String input)
    {return password.equals(input);}

    /**
     * Retrieves the current balance of the user.
     * @return The balance of the user as a floating-point value.
     */
    public float GetBalance() {
        return balance;
    }

    /** The username of the user. */
    private String username;
    /** The password of the user. */
    private String password;
    /** The name of the user. */
    private String name;
    /** The balance of the user. */
    private float balance;
    /** The account type of the user ("Seller" or "Customer"). */
    private final String userType;
}
```

Github Link of the Project:

https://github.com/hshaikh2020/COP4331ShoppingCartApp

Demo Link:

https://www.youtube.com/watch?v=yaforCYNZmA

Readme:

About Project:

The Shopping Cart Application is a Java-based desktop program designed to simulate an online shopping platform. It features distinct functionalities for both customers and sellers, such as managing user accounts, handling transactions, and organizing product inventories.

Prerequisites

-Prerequisites: -Java JDK 8 or newer installed on your system. -Access to the source code (either through Git clone or direct download). IDE like IntelliJ, Netbeans, VScode or any of your choice that is compatible to run java

Installation

Clone the Git repository or download the source code. bash Copy code git clone Navigate to the project directory.

Clone the repo

```
git clone https://github.com/hshaikh2020/COP4331ShoppingCartApp
```

Or you can manually download the Repo as Zip fle from Code -> Download as zip file

Now if you cloned it to your program after-clone, Go into the ShoppingCart.java file and run the program

Manual download

UNZIP the Folder to get COP4331ShoppingCartApp-main

next => open COP4331ShoppingCartApp-main folder to see ShoppingCartApp folder next=> open ShoppingCartApp folder to access target folder

next=> open target folder to access ShoppingCartApp-1.0-SNAPSHOT.jar

next=> now double click on ShoppingCartApp-1.0-SNAPSHOT.jar to open and run the shoppcart app.

Note: If the program gives an error from double clicking, try running it through command prompt instead.

In command prompt, navigate to the directory containing the ShoppingCartApp-1.0-SNAPSHOT.jar file

Run the command "java -jar ShoppingCartApp-1.0-SNAPSHOT.jar"

Key Features

Separate login and registration systems for customers and sellers.

Inventory Management: Sellers can add, update, or remove products from their inventory

Shopping Cart System: Customers can browse products, add them to a shopping cart, and proceed to checkout

User Account Management: Users can update personal information and manage account settings

(back to top)

Usage

Once you have successfully ran the app click on "Dont have an account ? Register here"

Fill in you details according if Seller , click seller and type in username and password and your name and click register

Then as seller you can start adding products to sell by clicking "Add new item"

Type in you product name, description, price and amount of stock and click iteam is on sale and click on add

you should be able to see your item in display inventory

Logout of account once you are done as seller

Now to Create a Customer account follow same steps as Step 2 but now choose Customer

Once you are logged in as customer you should see a list of products for sale

Go to User settings and section where it says "Add Funds" this is where we authorize our balance

type in the amount of money you want to add to your balance and hit save

now click go back , you should see you updated balance on top right of the app now from the Shopping menu choose the item you want to buy and click on it

You should be seeing a screen where it asks you how many items to add to cart

choose the amount of items and click add to cart

Now click on Shoppping Cart tab and click on complete transaction and it should show you cost so click done

The amount you paid should be deducted from you balance

Now logout from customer account

Log back in with your Seller credentials

You should be able to see the amount you paid as customer credited to your seller account