COP 4331 Object-Oriented Design & Programming

Fall 2023

Date: 10/28/2023

Report 3: Final Report

# *ItemHub*

Platform: SwingUI

Instructor: Dr. Ionut Cardei

Team Number: 5

Jamie Pham, Jason Lockett, Juan Arbelaez Ciro, Nicolas Uribe

# Application Requirements Feedback 1

- Modifications needed:
    - Add in project title (i.e. Store title), glossary and platform.
    - Write down explicitly what platform you will be using. Sing or Android/etc.
    - You need a UC where the seller reads about revenue, sales, profit.
    - Briefly explain functional specification as to how the application is expected to function and interact with users.
    - Write a glossary with the important domain concepts, like transaction, product, seller, etc.

# 1) Application Requirements:

- Write all application functional specifications in a **PDF file**.
    - Register for Customer and Seller:
        - The first window of the application will display "create an account" for the user, and there will be an input object for both a Username and Password and a button "Register".
    - Login for Customer and Seller:
        - A window the user is presented with after already registering for an account. There will be an input object for both a Username and Password and a button "Login".
    - View Shopping Cart:
        - Top right shopping cart icon, once clicked, shows a list of items the users added. Displays total cart price underneath the icon.
    - Search Item:
        - A search field in the top middle of the main page.
    - Add item to cart:
        - The items displayed on the main page will have a button that once clicked, adds an item to the user's shopping cart.
    - Delete item in cart:
        - Inside the shopping cart display, there will be a small red trash can icon that will delete an item completely from the cart.
    - Inventory:
        - (Logged in as seller) There will be an inventory icon on the main page. Once clicked the page will refresh and show the items in inventory. There will be options to add items to inventory and view item descriptions with more options to update information.
    - Display product information:

        - Items that are displayed from search will only show the name and price. Once the user clicks on the item icon,  a window will pop up to show the product information.
    - Change item count in cart:
        - Inside the shopping cart, there will be an input object where the customer will be able to adjust the quantity for each item with plus or minus buttons or a text input to insert a specific number.
    - Checkout button in shopping cart:
        - User is inside the shopping cart. Below the list of items is a button that is called “checkout”. This button leads to the checkout window.
    - Display items in checkout window:
        - In the checkout window, a summary of the items in the shopping cart will be displayed.
    - Have payment information fill in form:
        - In the checkout window there is also a fill in form that will request the payment information from the user.
    - Performance Dashboard:
        - This window is displayed when the Seller logs in. There will be three big columns presenting the seller with their total revenue, sales, and profit.

- Write Use Cases (essential first, then detailed).

  **Specific Use Cases for Option#1: the Shopping Cart:**
  Write at least the following use cases. Add more if necessary.

    - User Logs In
    - Customer Adds Items to Shopping Cart
    - Customer Reviews Product Details
    - Customer Reviews/Updates Shopping Cart
    - Customer Checks Out
    - Seller Reviews/Updates Inventory
    - Seller Adds New Product
    - Seller Checks Revenue, Sales, and Profit

    - <u>Use Cases:</u>
        1) Registering/logging in as Customer/Seller:
           Actors: Customer/Seller

1. Customer/Seller launches application.
2. Application asks the user if they would like to login or register a new account.
3. Customer/Seller selects register/login for customer/seller.
4. Customer/Seller enters Username and Password.
5. Customer/Seller is taken to the respective account home page.

1.1) **Variation** - Entering in wrong credentials when logging in:

- In step 4, the customer/seller enters invalid username or password.
- A pop-up message lets the customer/seller know that the information entered is incorrect.
- Customer/Seller is returned to the register/login page.

1.2) **Variation** - Logging in as wrong type of user:

- From step 3, the customer selects login as seller, or the seller selects login as customer accidentally.
- The Customer/Seller enters Username and Password
- A pop-up message states "The information you have entered doesn't match the account type selected. Make sure you have selected the right account type."
- Customer/Seller is returned to the account selection page.

2) Reviewing Product Information:

Actors: Customer

1. Customer logs in.
2. Customer browses/scrolls through product list.
3. Customer selects a preferred product.
4. Pop-up window displays product icons and further information.

3) Adding Items to Cart:

Actors: Customer

1. Customer logs in.
2. Customer browses/scrolls through product list.
3. Customer clicks "Add to Cart" underneath the product icon.
4. A pop-up message lets the customer know that the product was successfully added to the cart.

3.1) **Variation** - Adding an out of stock product to cart:

- In step 3, the customer attempts to add a product to the cart.
- A pop-up message appears letting the customer know that the product is out of stock.
- The customer dismisses the message and gets redirected to the product list.

4) Updating Shopping Cart Items:

Actors: Customer

1. Customer selects the shopping cart icon in the upper right corner.
2. Shopping cart information is displayed.
3. For each item, the following options are available:
    - Delete from cart.
    - Add/subtract from the quantity.
4. Customer edits shopping cart.

5) Customer Purchases Item:

   Actors: Customer

   1. Customer selects the shopping cart icon in the upper right corner.
   2. Shopping cart information is displayed.
   3. Customer selects the checkout button in the bottom right corner.
   4. Window displays the payment information form.
   5. Customer fills in the form and hits the continue button in the bottom right corner.
   6. Window displays a confirmation page with filled out forms and a summary of the selected items.
   7. Customer selects “Confirm purchase” displayed at the bottom of the page.
   8. Window displays “Successful Purchase!”

   5.1) **Variation** - Entering in wrong credit card format:

   - From Step 4, Customer fills in the payment information form and hits the continue button in the bottom right corner.
   - System detects incorrect or incomplete credit card information.
   - Window displays an error message: “Invalid credit card details. Please check and enter again.”
   - Customer reviews the credit card details and corrects the information.
   - Continue from Step 6

6) Seller Checks Revenue, Sales, and profit:

   Actors: Seller

   1. Seller launches application.
   2. Seller selects login for seller.
   3. Seller enters login credentials and hits login.
   4. Seller is taken to the main dashboard where he/she is presented with their data analysis of Revenue, Sales, and profit in three different columns.

7) Seller Reviews/Updates Inventory:

Actors: Seller

1. Seller launches application.
2. Seller selects login for seller.
3. Seller enters login credentials and hits login.
4. Seller selects the inventory icon on the upper right corner of the seller dashboard.
5. A window displays the information for the items in the inventory.
6. Seller selects a product.
7. Pop-up window displays product info that can be updated.
8. Seller makes necessary changes to product information.

7.1) **Variation** - Entering in negative quantity when restocking product:

- From step 5, Seller selects a product.
- Pop-up window displays product info that can be updated.
- Seller enters a non-positive number for inventory count or price.
- System detects invalid entries and displays “Please enter a positive number for inventory count/price.”
- Seller enters valid numbers.
- Seller saves the updated product info.

8) Seller Adds New Product:

Actors: Seller

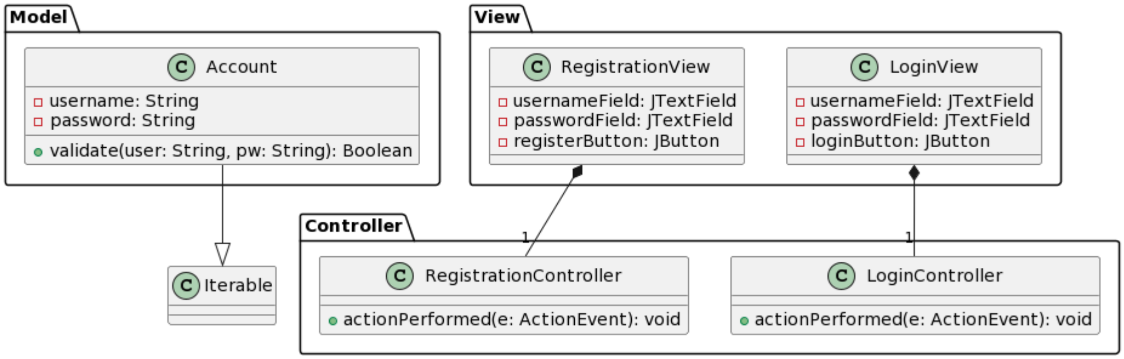
1. Seller selects inventory icon in the upper right corner.
2. A window displays the information for items in the inventory.
3. Seller selects the “plus” icon at the bottom of the page to add a new product.
4. Pop-up window displays a new product information form.
5. Seller enters required information for new product.
6. Seller selects “Add new product”.
7. A window displays updated information for the items in inventory.
    - Includes new products.

# Glossary

- Product:
  - An item available for purchase through the shopping cart application, such as merchandise, services, or digital goods.
- Seller:
  - An entity or individual that offers products for sale through the shopping cart application. Sellers could be individuals, businesses, or organizations.
- Customer:
  - An individual or entity using the shopping cart application to browse, select, and purchase products.
- Shopping Cart:
  - A virtual container within the application where customers can add and manage selected products before proceeding to checkout.
- Checkout:
  - The process where customers finalize their purchases, providing payment and shipping information before completing the transaction.
- Inventory:
  - A record of available products and their quantities that the shopping cart application manages to ensure accurate order fulfillment.
- Performance Dashboard:
  - A graphical interface that provides sellers a concise and visual representation of revenue, sales, and profit, to help them evaluate and manage their business performance.

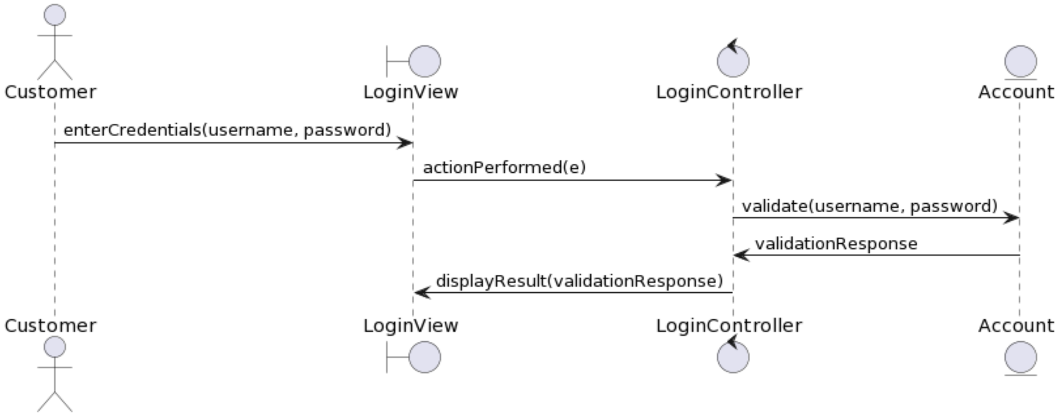
# 2) Design Specification:

## Class Diagram for Use Case 1:

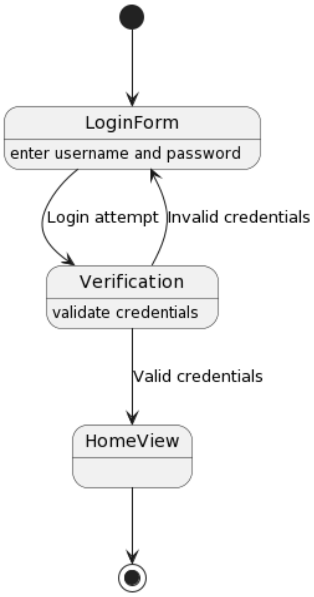


Model
Account
- username: String
- password: String
+ validate(user: String, pw: String): Boolean
Iterable
View
RegistrationView
- usernameField: JTextField
- passwordField: JTextField
- registerButton: JButton
LoginView
- usernameField: JTextField
- passwordField: JTextField
- loginButton: JButton
Controller
1
RegistrationController
+ actionPerformed(e: ActionEvent): void
1
LoginController
+ actionPerformed(e: ActionEvent): void

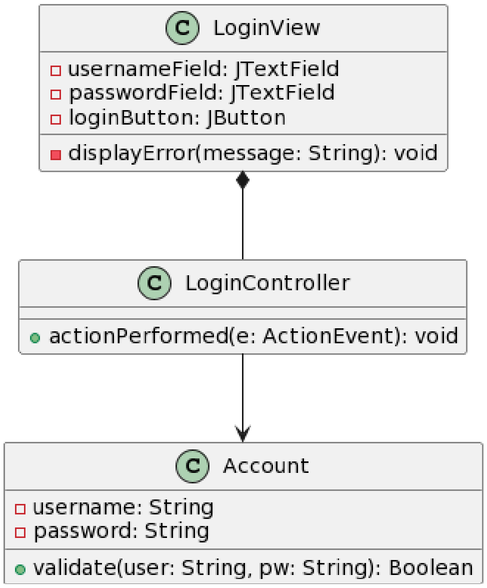
## Sequence Diagram for Use Case 1:



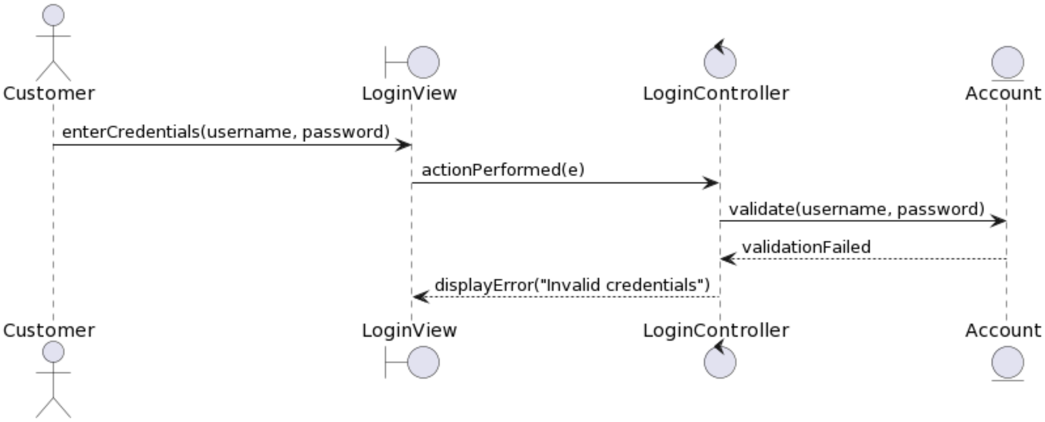
## State Diagram for Use Case 1:



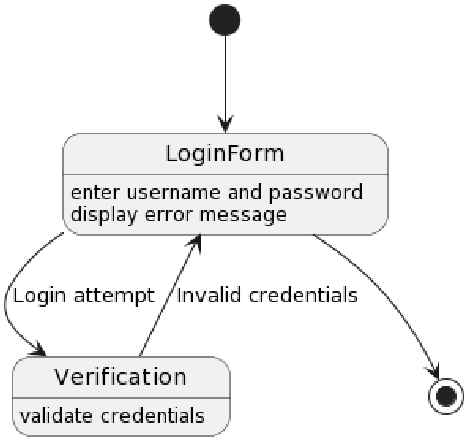
## Class Diagram for Use Case 1.1 (Variant):


C LoginView
usernameField: JTextField
passwordField: JTextField
loginButton: JButton
displayError(message: String): void
C LoginController
actionPerformed(e: ActionEvent): void
C Account
username: String
password: String
validate(user: String, pw: String): Boolean

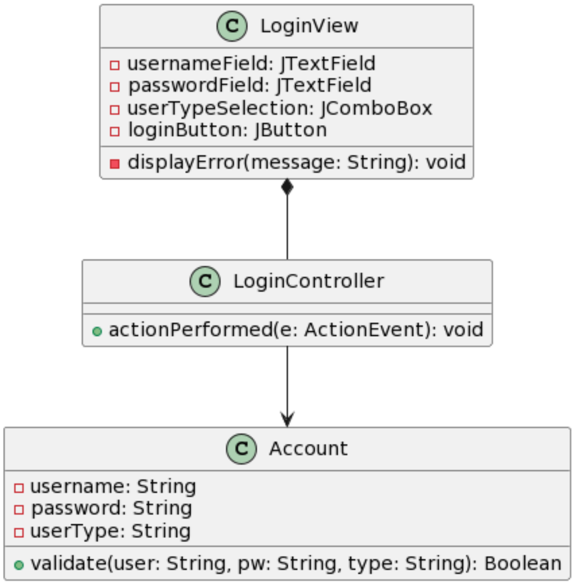
## Sequence Diagram for Use Case 1.1 (Variant):



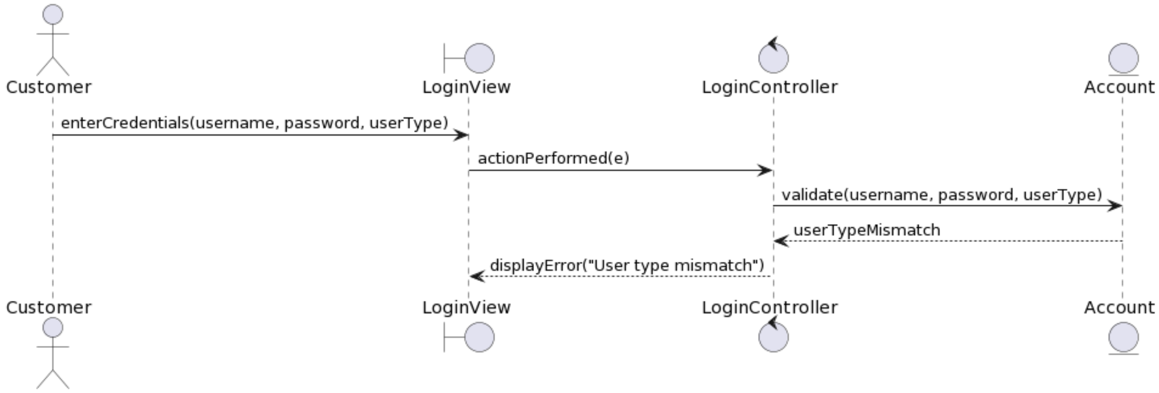
## State Diagram for Use Case 1.1 (Variant):



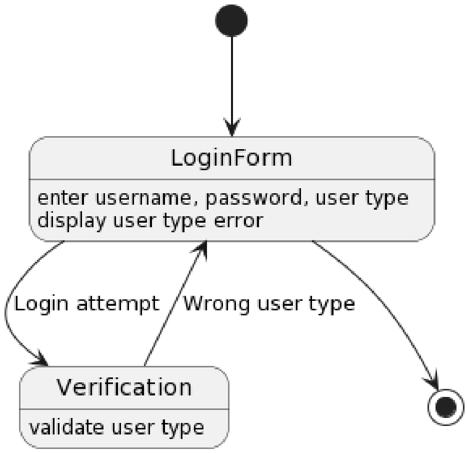
## Class Diagram for Use Case 1.2 (Variant):


C
LoginView
usernameField: JTextField
passwordField: JTextField
userTypeSelection: JComboBox
loginButton: JButton
displayError(message: String): void
C
LoginController
actionPerformed(e: ActionEvent): void
C
Account
username: String
password: String
userType: String
validate(user: String, pw: String, type: String): Boolean

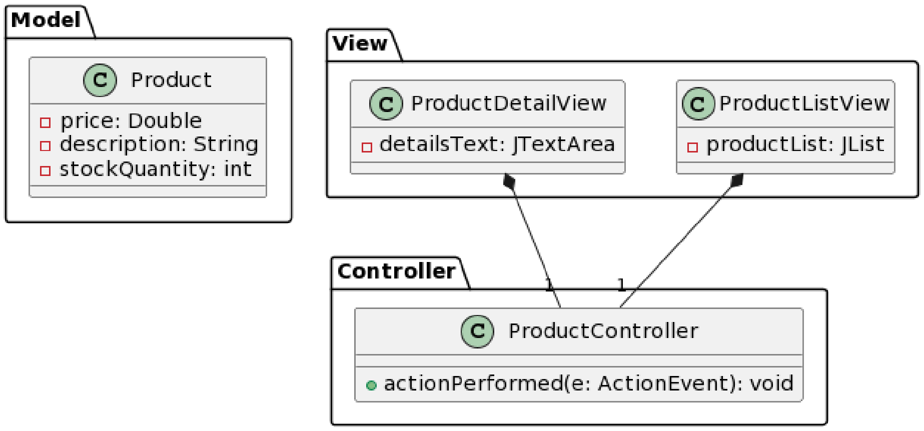
## Sequence Diagram for Use Case 1.2 (Variant):


Customer
LoginView
LoginController
Account
enterCredentials(username, password, userType)
actionPerformed(e)
validate(username, password, userType)
userTypeMismatch
displayError("User type mismatch")
Customer
LoginView
LoginController
Account

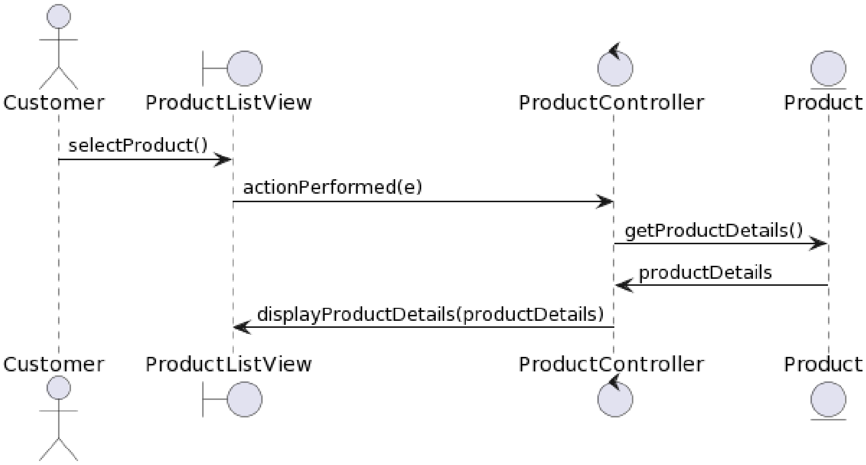
## State Diagram for Use Case 1.2 (Variant):



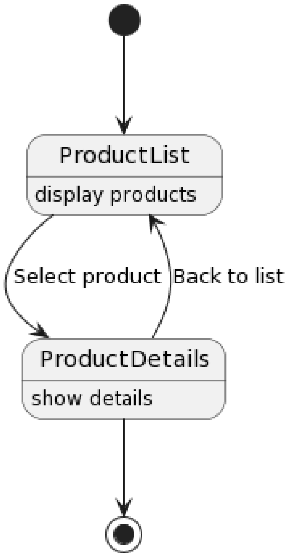
## Class Diagram for Use Case 2:



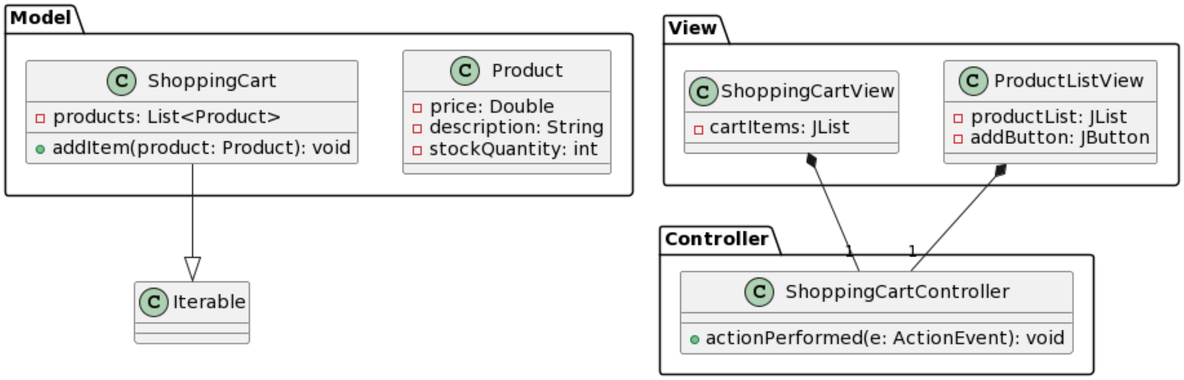
## Sequence Diagram for Use Case 2:



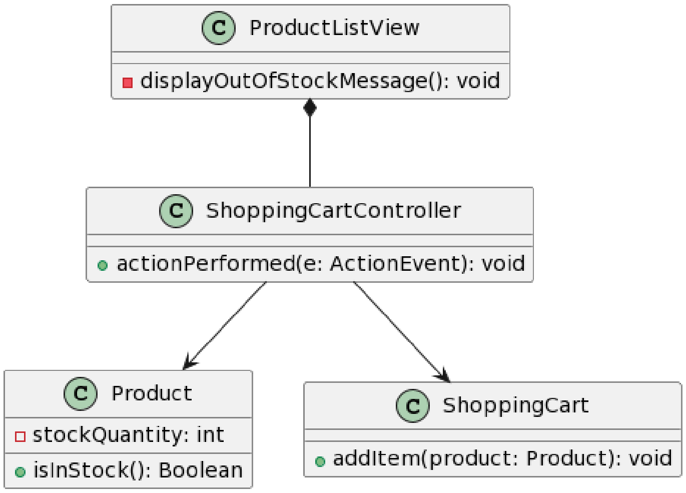
## State Diagram for Use Case 2:



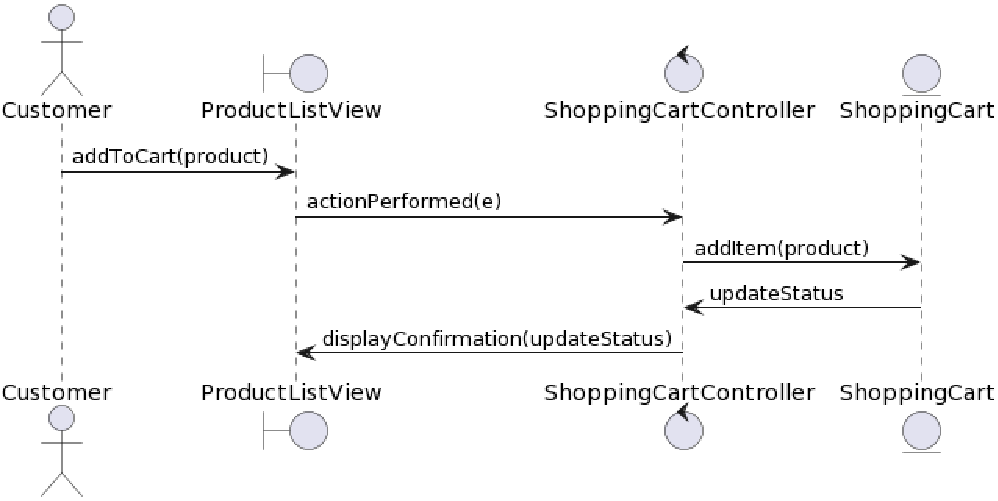
## Class Diagram for Use Case 3:



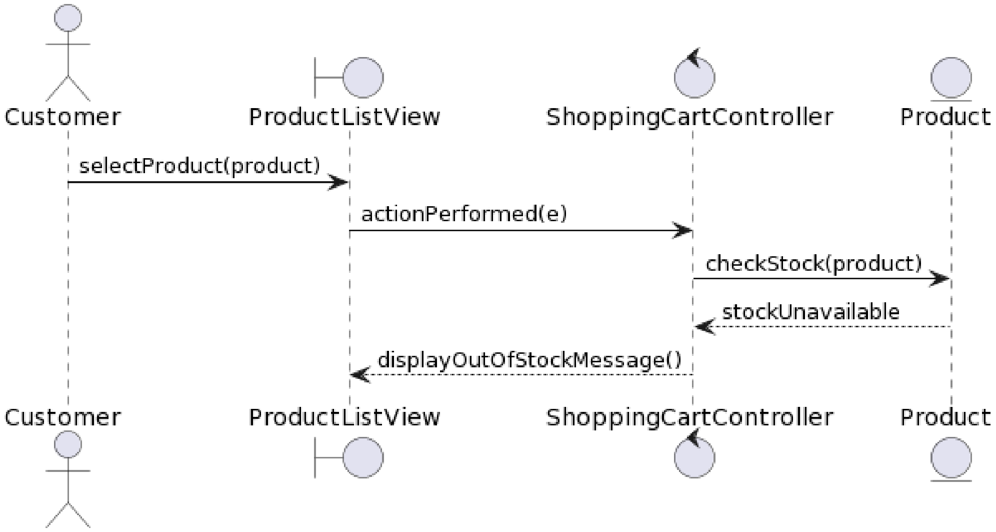
## Class Diagram for Use Case  3.1 (Variant):


C ProductListView
■ displayOutOfStockMessage(): void
C ShoppingCartController
● actionPerformed(e: ActionEvent): void
C Product
□ stockQuantity: int
● isInStock(): Boolean
C ShoppingCart
● addItem(product: Product): void

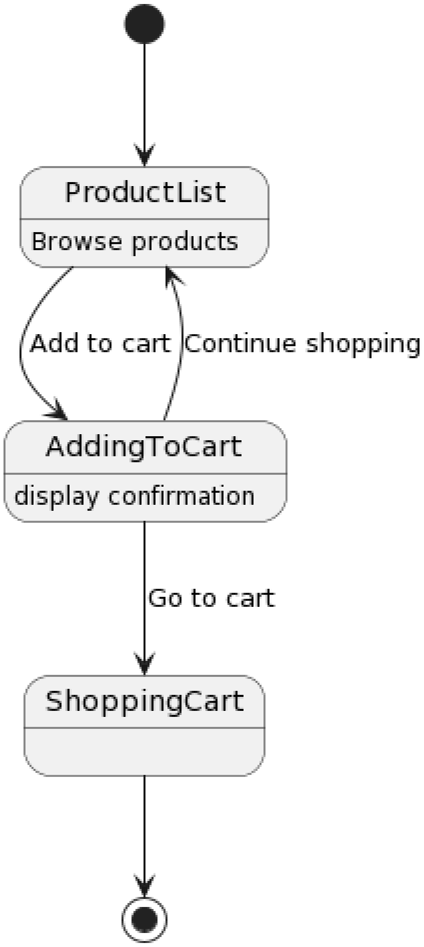
## Sequence Diagram for Use Case 3:



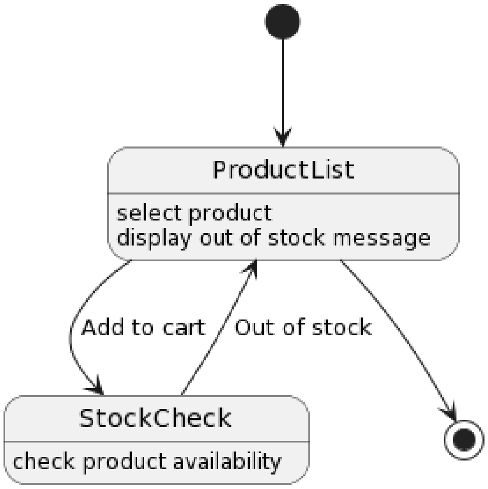
## Sequence Diagram for Use Case 3.1 (Variant):



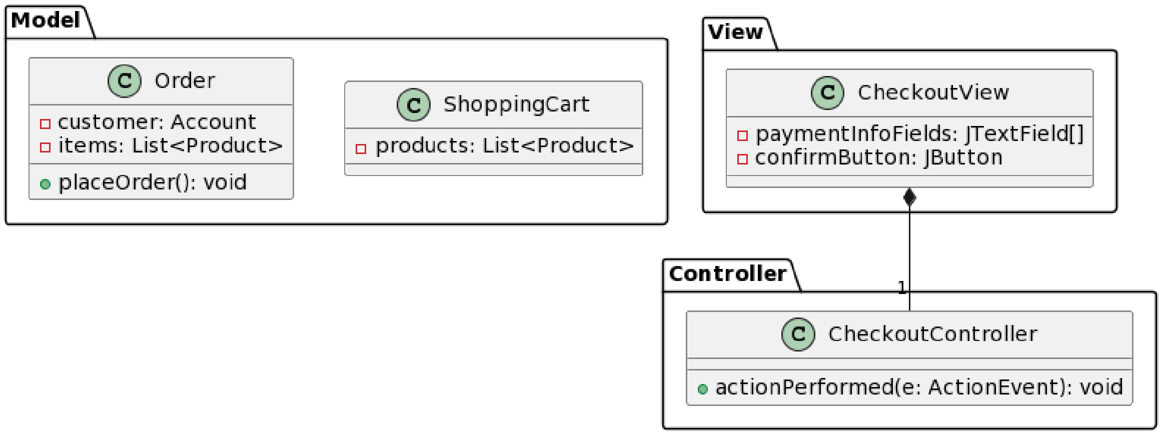
## State Diagram for Use Case 3:



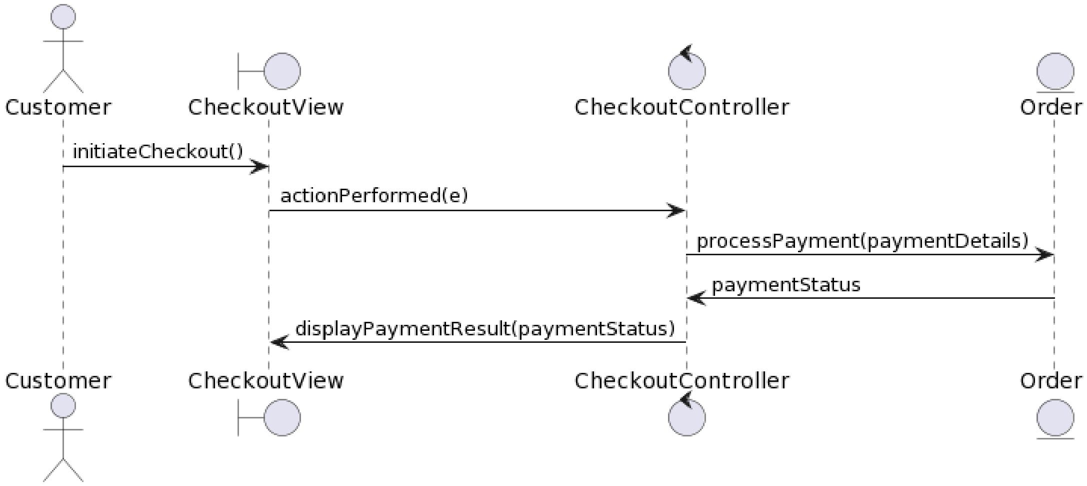
## State Diagram for Use Case 3.1 (Variant):



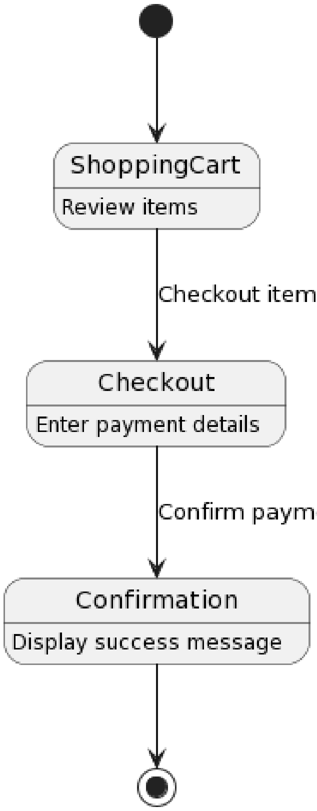
## Class Diagram for Use Case 4:



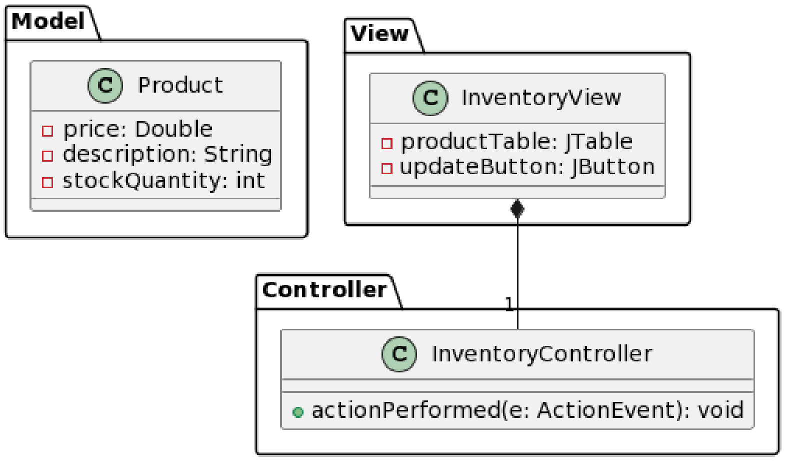
## Sequence Diagram for Use Case 4:



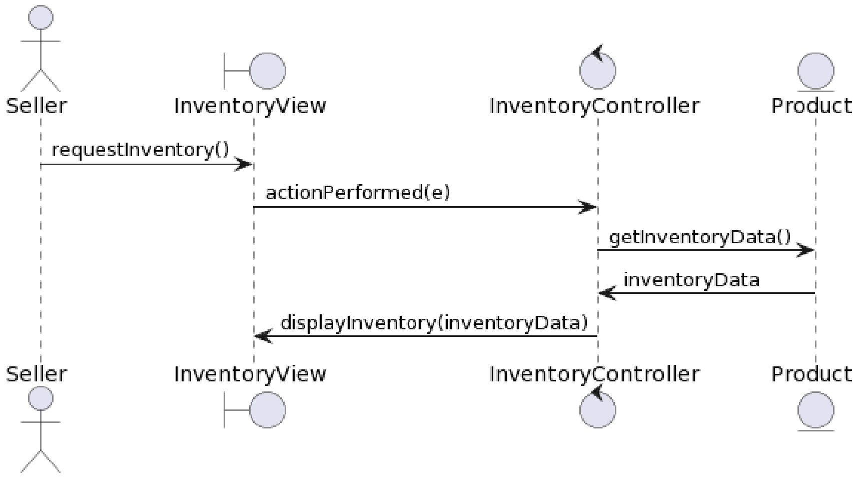
## State Diagram for Use Case 4:



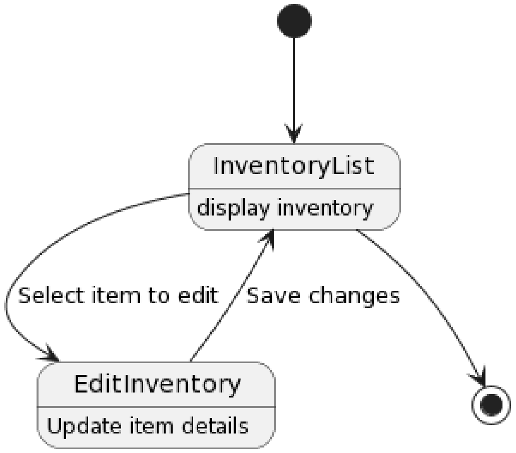
## Class Diagram for Use Case 5:



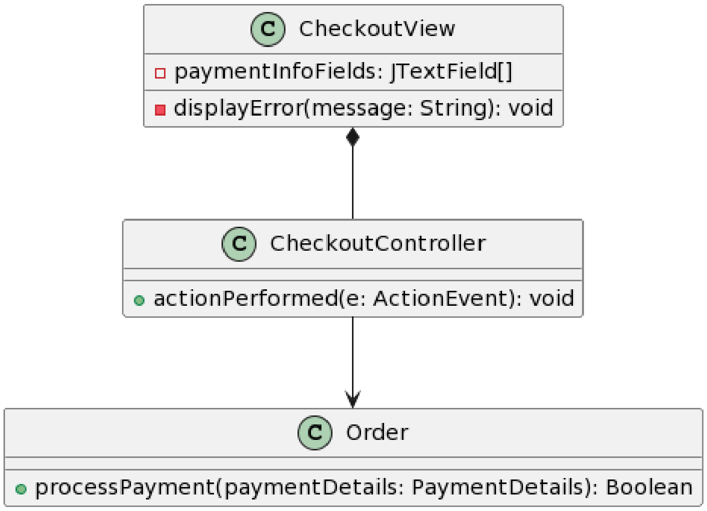
## Sequence Diagram for Use Case 5:



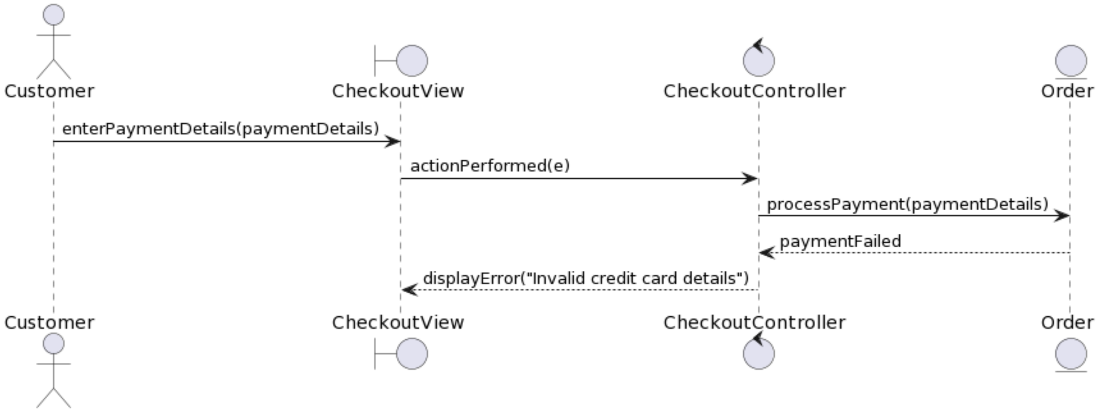
## State Diagram for Use Case 5:



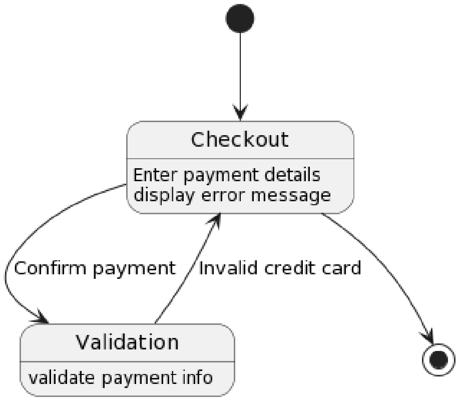
## Class Diagram for Use Case 5.1 (Variant):


C CheckoutView
paymentInfoFields: JTextField[]
displayError(message: String): void
C CheckoutController
actionPerformed(e: ActionEvent): void
C Order
processPayment(paymentDetails: PaymentDetails): Boolean

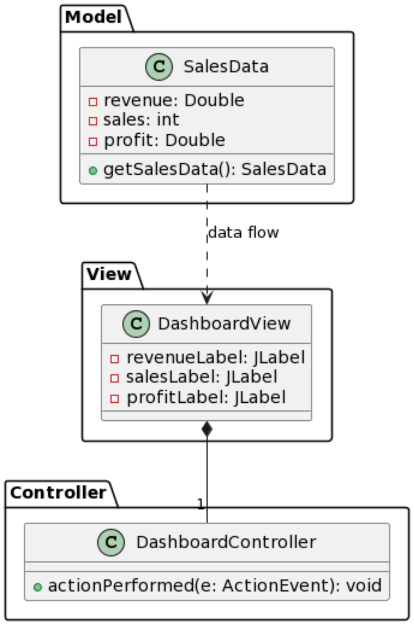
## Sequence Diagram for Use Case 5.1 (Variant):



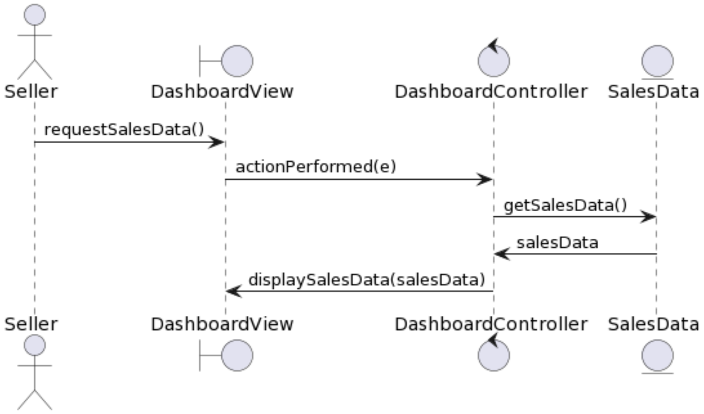
## State Diagram for Use Case 5.1(Variant):



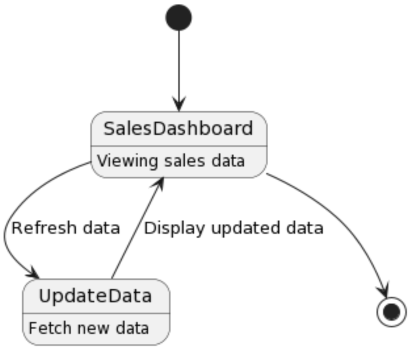
## Class Diagram for Use Case 6:



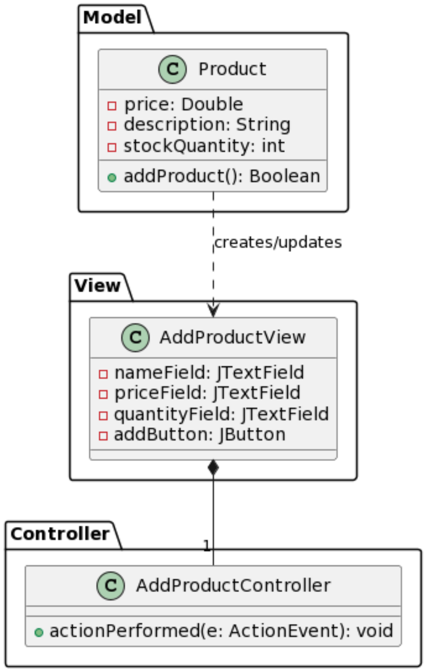
## Sequence Diagram for Use Case 6:



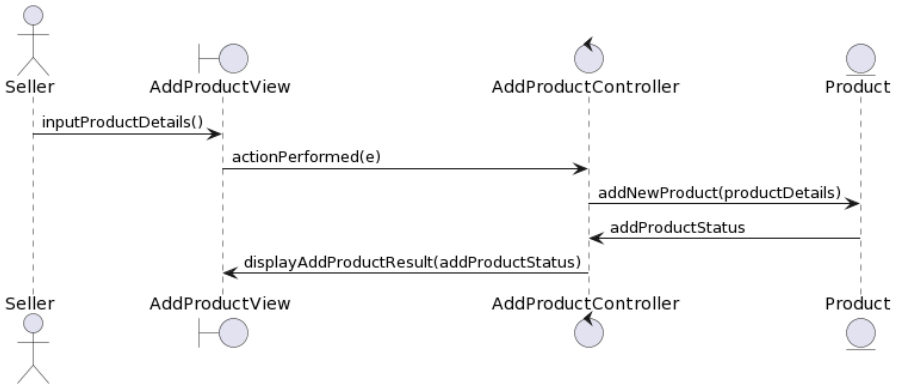
## State Diagram for Use Case 6:



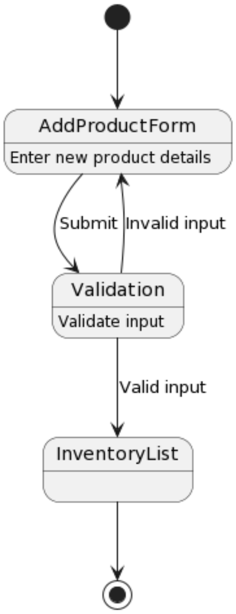
## Class Diagram for Use Case 7:



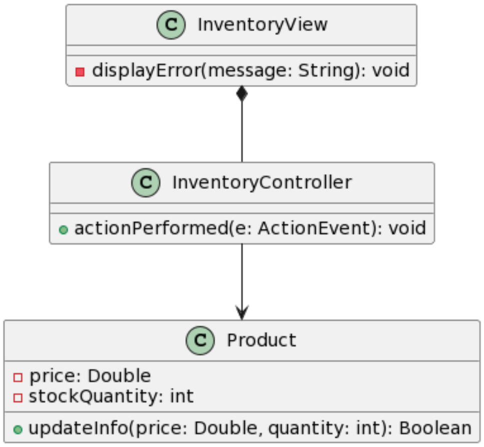
## Sequence Diagram for Use Case 7:



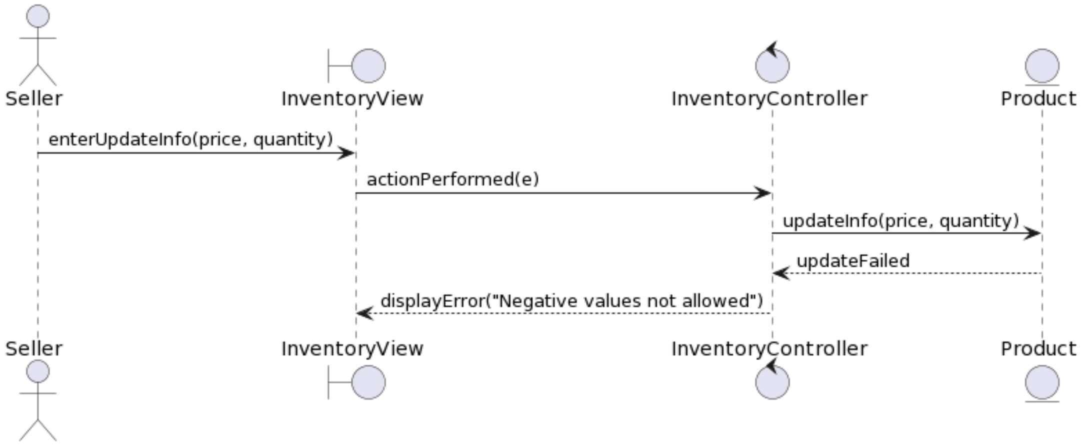
## State Diagram for Use Case 7:



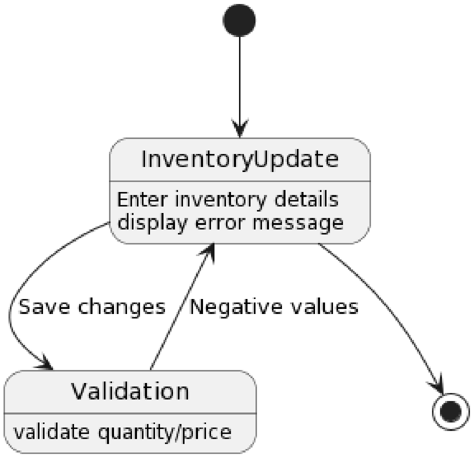
## Class Diagram for Use Case 7.1 (Variant):


InventoryView
displayError(message: String): void
InventoryController
actionPerformed(e: ActionEvent): void
Product
price: Double
stockQuantity: int
updateInfo(price: Double, quantity: int): Boolean

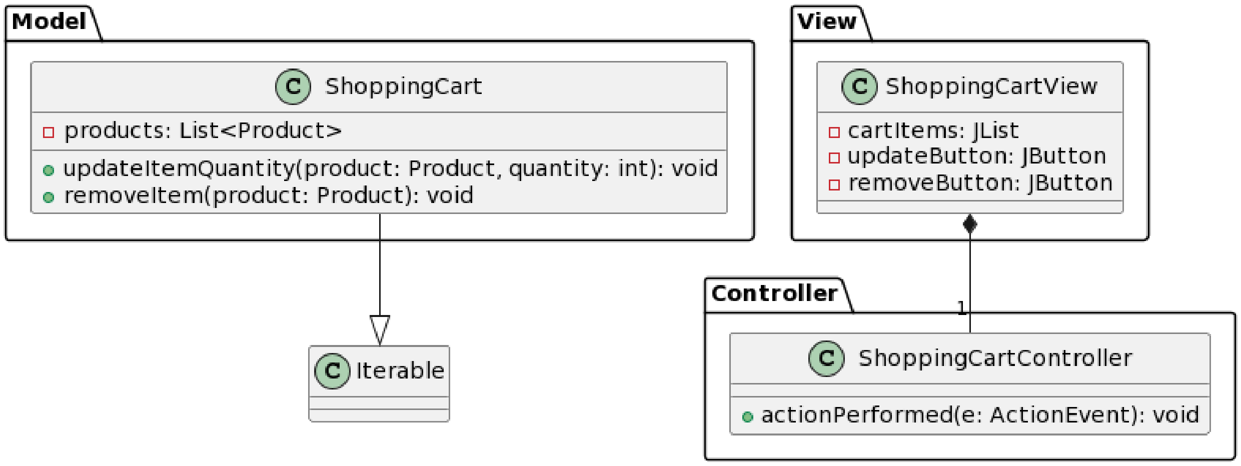
## Sequence Diagram for Use Case 7.1 (Variant):



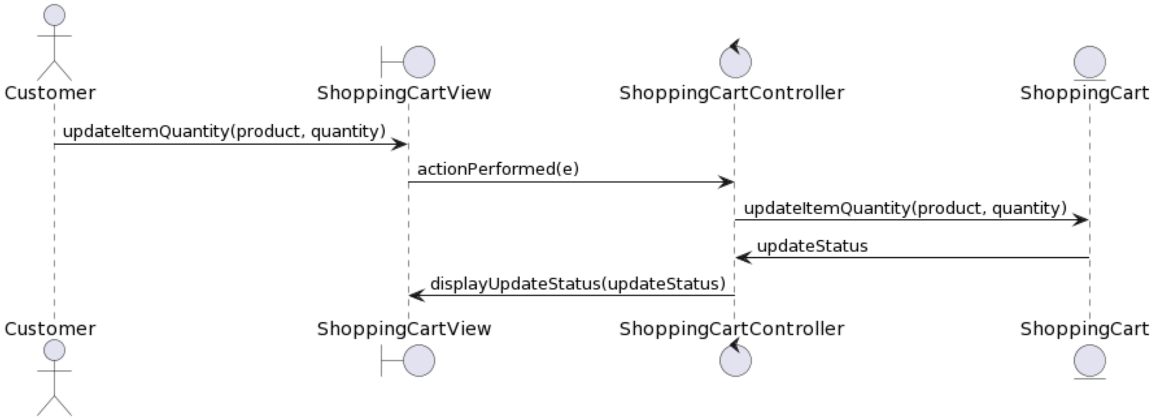
## State Diagram for Use Case 7.1 (Variant):



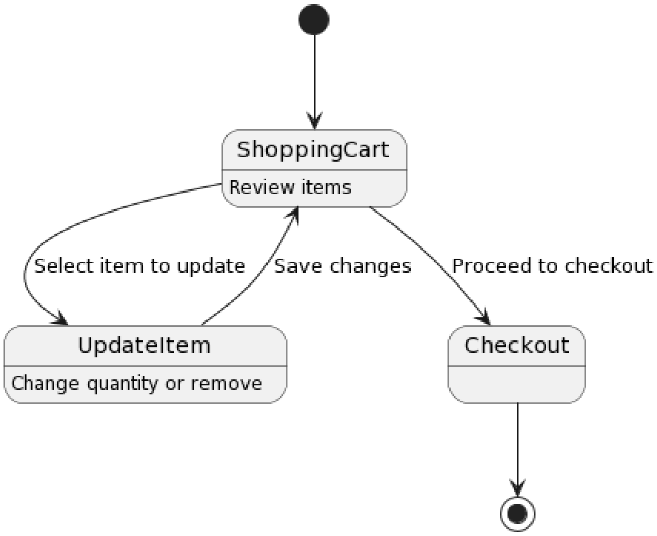
## Class Diagram for Use Case 8:


Model
C ShoppingCart
□ products: List<Product>
● updateItemQuantity(product: Product, quantity: int): void
● removeItem(product: Product): void
C Iterable
View
C ShoppingCartView
□ cartItems: JList
□ updateButton: JButton
□ removeButton: JButton
1
Controller
C ShoppingCartController
● actionPerformed(e: ActionEvent): void

## Sequence Diagram for Use Case 8:



## State Diagram for Use Case 8:



# *Source Code*

## *Application.java*

```java
/**
 * Shopping Cart Application
 * Main class that initializes the login view and the controller
 * Also displays the login view.
 */
public class Application {
    public static void main(String[] args) {
        LoginView view = new LoginView();
        LoginController controller = new LoginController(view);
        view.setVisible(true);
    }
}
```

## *LoginController.java*

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;

/**
 * Login controller that manages the user authentication and registration of the application.
 */
public class LoginController {
    private LoginView loginView;
    private Map<String, User> userDatabase;

    /**
     * LoginController constructor with the specified LoginView.
     * @param loginView view associated with this controller.
     */
    public LoginController(LoginView loginView) {
        this.loginView = loginView;
        this.userDatabase = new HashMap<>();

        this.loginView.addLoginListener(new LoginListener());
        this.loginView.addRegisterListener(new RegisterListener());
    }

    /**
     * Login listener that implements an actionListener to handle login attempts.
     * The Observer pattern used when listening to the login events.
     */
    class LoginListener implements ActionListener {
        /**
         * An action that is invoked when the login button is clicked.
         * @param e is the ActionEvent that represents the button click.
         */
        public void actionPerformed(ActionEvent e) {
            String username = loginView.getUsername();
            String password = loginView.getPassword();
            String userType = loginView.getUserType();

            if (userDatabase.containsKey(username) &&
userDatabase.get(username).getPassword().equals(password)) {
                if ("Customer".equals(userDatabase.get(username).getUserType())) {
                    loginView.displayErrorMessage("Login Successful as Customer!");
                    ProductView productView = new ProductView();
                    productView.setVisible(true);
                    loginView.dispose();
                } else if ("Seller".equals(userDatabase.get(username).getUserType())) {
                    loginView.displayErrorMessage("Login Successful as Seller!");
                    SellerView sellerView = new SellerView();
                    sellerView.setVisible(true);
                    loginView.dispose();
                } else {
                    loginView.displayErrorMessage("Invalid user type!");
                }
            } else {
                loginView.displayErrorMessage("Invalid credentials!");
            }
        }
    }

    /**
```

```
     * Registration listener that implements an actionListener to handle user registration.
     * The Observer pattern is used when listening to registration events.
     */
    class RegisterListener implements ActionListener {

        /**
         * An action that is Invoked when the register button is clicked.
         * @param e is the ActionEvent that represents the button click.
         */
        public void actionPerformed(ActionEvent e) {
            String username = loginView.getUsername();
            String password = loginView.getPassword();
            String userType = loginView.getUserType();

            if (!userDatabase.containsKey(username)) {
                userDatabase.put(username, new User(username, password, userType));
                loginView.displayErrorMessage("Registration Successful as " + userType + "!");
            } else {
                loginView.displayErrorMessage("Username already exists!");
            }
        }
    }
}
```

# *LoginView.java*

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

/**
 * The LoginView class that represents the login interface of the application.
 */
public class LoginView extends JFrame {
    private JTextField usernameField = new JTextField(10);
    private JPasswordField passwordField = new JPasswordField(10);
    private JComboBox<String> userTypeComboBox = new JComboBox<>(new
String[]{"Customer", "Seller"});
    private JButton loginButton = new JButton("Login");
    private JButton registerButton = new JButton("Register");

    /**
     * LoginView window constructor that implements UI components for username, password,
     * user type, login button, and registration button.
     */
    public LoginView() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 200);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(10, 10));

        JPanel formPanel = new JPanel(new GridLayout(4, 2, 10, 10));
        formPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        formPanel.add(new JLabel("User Type:"));
        formPanel.add(userTypeComboBox);
        formPanel.add(new JLabel("Username:"));
    formPanel.add(usernameField);
    formPanel.add(new JLabel("Password:"));
    formPanel.add(passwordField);

    JPanel buttonPanel = new JPanel(new FlowLayout());
    styleButton(loginButton, new Color(100, 149, 237));
    styleButton(registerButton, new Color(30, 144, 255));
    buttonPanel.add(loginButton);
    buttonPanel.add(registerButton);

    add(formPanel, BorderLayout.CENTER);
    add(buttonPanel, BorderLayout.SOUTH);
}

/**
 * Styling for the JButton
 * @param button  is the JButton to style.
 * @param color  is the background color for the button.
 */
private void styleButton(JButton button, Color color) {
    button.setFont(new Font("Arial", Font.PLAIN, 14));
    button.setBackground(color);
    button.setForeground(Color.WHITE);
    button.setBorderPainted(false);
    button.setOpaque(true);
}

/**
 * Getters for Username, Password, and UserType.
 * @return  the entered Username, Password, and UserType.
 */
```

```java
    public String getUsername() {
        return usernameField.getText();
    }
    public String getPassword() {
        return new String(passwordField.getPassword());
    }
    public String getUserType() {
        return userTypeComboBox.getSelectedItem().toString();
    }

    /**
     * ActionListener for the login and registration buttons.
     */
    void addLoginListener(ActionListener listenForLoginButton) {
        loginButton.addActionListener(listenForLoginButton);
    }
    void addRegisterListener(ActionListener listenForRegisterButton) {
        registerButton.addActionListener(listenForRegisterButton);
    }

    /**
     * Display for an error message with a specified error.
     * @param errorMessage  is the error message that will be displayed.
     */
    void displayErrorMessage(String errorMessage) {
        JOptionPane.showMessageDialog(this, errorMessage);
    }
}
```

# *Product.java*

```
/**
 * Product class for the application.
 */
public class Product {
    private String name;
    private double price;
    private String description;
    private int quantity;
    private double cost;

    /**
     * Product constructor that creates a new product with the specified attributes.
     * @param name        is the name of the product. Can't be null.
     * @param cost        is the cost of the product. Can't be negative.
     * @param price       is the selling price of the product. Can't be negative.
     * @param description is a description of the product. Can't be null
     * @param quantity    is the quantity of the product in stock. Can't be negative.
     */
    public Product(String name, double cost, double price, String description, int quantity) {
        this.name = name;
        this.cost = cost;
        this.price = price;
        this.description = description;
        this.quantity = quantity;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
}

public double getCost() {
    return cost;
}
public void setCost(double cost) {
    this.cost = cost;
}

public double getPrice() {
    return price;
}
public void setPrice(double price) {
    this.price = price;
}

public int getQuantity() {
    return quantity;
}
public void setQuantity(int quantity) {
    this.quantity = quantity;
}

public String getDescription() {
    return description;
}
public void setDescription(String description) {
    this.description = description;
}

/**
```

```
     * Turns the product information into an array of objects for displaying purposes.
     * @return An array of objects that represent the products information.
     */
    public Object[] toRow() {
        return new Object[]{name, price, quantity};
    }
}
```

# *ProductData.java*

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

/**
 * Class that provides access to product data for the application.
 * Singleton pattern because there is a single instance of the product data for the entire
application.
 */
public class ProductData {
    private static final List<Product> products = new ArrayList<>();
    static {
        products.add(new Product("Water Bottle", 5.00, 10.00, "This is a 16oz water bottle", 10));
        products.add(new Product("Teddy Bear", 10.00, 70.00, "This is a light brown Teddy Bear",
5));
        products.add(new Product("Sunglasses", 5.00, 20.00, "This is a simple pair of black
sunglasses", 20));
        products.add(new Product("Running Shoes", 50.00, 100.00, "This is a pair of Running
Shoes", 10));
        products.add(new Product("Monitor", 200.00, 1000.00, "This is a simple 27 inch monitor",
3));
    }

    /**
     * Add a new product to the product list.
     * Get a list of available products from the product list.
     * Update the quantity of a single product from the product list.
     * Products can't be null and the quantity must be positive to increase, and negative for
decrease.
     */
    public static void addProduct(Product product) {
        products.add(product);
```

```
}
public static List<Product> getProducts() {
    return new ArrayList<>(products);
}
public static void updateProductQuantity(Product product, int quantityChange) {
    for (Product p : products) {
        if (p.equals(product)) {
            int newQuantity = Math.max(p.getQuantity() + quantityChange, 0);
            p.setQuantity(newQuantity);
            break;
        }
    }
}

private static double totalRevenue = 0.0;
private static int totalSales = 0;
private static double totalProfit = 0.0;

/**
 * A constructor that records a sale and updates the total revenue, total sales, and total profit.
 * @param product  is the product being sold. Can't be null.
 * @param quantity is the quantity sold. Needs to be greater than 0.
 */
public static void recordSale(Product product, int quantity) {
    double revenueFromSale = product.getPrice() * quantity;
    double costOfSale = product.getCost() * quantity;
    totalRevenue += revenueFromSale;
    totalProfit += (revenueFromSale - costOfSale);
    totalSales += quantity;
}
public static double getTotalRevenue() {
        return totalRevenue;
    }
    public static int getTotalSales() {
        return totalSales;
    }
    public static double getTotalProfit() {
        return totalProfit;
    }
}
```

# *ProductView.java*

```
import javax.swing.*;
import javax.swing.plaf.ColorUIResource;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.ArrayList;

/**
 * ProductView class that represents the product catalog view in the application.
 * Handling the display of products, shopping cart management, and checkout process.
 */
public class ProductView extends JFrame {
    private List<Product> products;
    private JPanel productPanel;
    private ShoppingCart shoppingCart;
    private JTextField searchField;
    private JButton cartButton;
    private JButton logoutButton;

    /**
     * ProductView constructor that creates a window with a product catalog and navigation bar.
     */
    public ProductView() {
        setTitle("Product Catalog");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        UIManager.put("ScrollBar.width", 15);
        UIManager.put("ScrollBar.thumb", new ColorUIResource(new Color(100, 149, 237)));
        UIManager.put("ScrollBar.thumbHighlight", new
ColorUIResource(Color.LIGHT_GRAY));
        UIManager.put("ScrollBar.thumbDarkShadow", new
ColorUIResource(Color.DARK_GRAY));
        UIManager.put("ScrollBar.thumbLightShadow", new ColorUIResource(Color.GRAY));
        UIManager.put("ScrollBar.track", new ColorUIResource(new Color(240, 240, 240)));

        Timer refreshTimer = new Timer(5000, e -> refreshProducts());
        refreshTimer.start();

        shoppingCart = new ShoppingCart();
        productPanel = new JPanel();
        productPanel.setLayout(new GridLayout(0, 2, 10, 10));
        productPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        JPanel navBar = new JPanel(new BorderLayout(10, 10));
        JLabel storeName = new JLabel("My Store", JLabel.CENTER);
        storeName.setFont(new Font("Arial", Font.BOLD, 18));
        navBar.add(storeName, BorderLayout.WEST);

        searchField = new JTextField();
        navBar.add(searchField, BorderLayout.CENTER);

        JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        cartButton = new JButton(new ImageIcon("ShoppingCart/Images/SC.png"));
        styleButton(cartButton, new Color(100, 149, 237), Color.WHITE);
        cartButton.addActionListener(new ActionListener() {
            @Override
```

```
    public void actionPerformed(ActionEvent e) {
        showShoppingCart();
    }
});
rightPanel.add(cartButton);

logoutButton = new JButton("Logout");
styleButton(logoutButton, new Color(100, 149, 237));
logoutButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int response = JOptionPane.showConfirmDialog(ProductView.this,
                "Are you sure you want to log out?", "Confirm Logout",
                JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
        if (response == JOptionPane.YES_OPTION) {
            dispose();
            LoginView loginView = new LoginView();
            LoginController loginController = new LoginController(loginView);
            loginView.setVisible(true);
        }
    }
});
rightPanel.add(logoutButton);
navBar.add(rightPanel, BorderLayout.EAST);

add(navBar, BorderLayout.NORTH);

initializeProducts();
displayProducts();

JScrollPane scrollPane = new JScrollPane(productPanel);
```

```
    scrollPane.setBorder(BorderFactory.createEmptyBorder());
    add(scrollPane, BorderLayout.CENTER);

    setLocationRelativeTo(null);
}

private void styleButton(JButton button, Color color) {
    button.setFont(new Font("Arial", Font.PLAIN, 14));
    button.setBackground(color);
    button.setForeground(Color.WHITE);
    button.setBorderPainted(false);
    button.setOpaque(true);
}

/**
 * Product catalog in the view is refreshed with the updated products.
 */
private void refreshProducts() {
    List<Product> updatedProducts = ProductData.getProducts();
    Map<String, Product> productMap = new HashMap<>();
    for (Product p : products) {
        productMap.put(p.getName(), p);
    }
    for (Product p : updatedProducts) {
        if (!productMap.containsKey(p.getName())) {
            products.add(p);
        }
    }
    displayProducts();
    revalidate();
    repaint();
```

```java
}

/**
 * Initialize the product catalog with products.
 */
private void initializeProducts() {
    products = ProductData.getProducts();
}

/**
 * Styling a JButton with a specified background color and font.
 * @param button          is the JButton to style.
 * @param backgroundColor is the background color for the button.
 * @param textColor       is the text color for the button.
 */
private void styleButton(JButton button, Color backgroundColor, Color textColor) {
    button.setFont(new Font("Arial", Font.BOLD, 14));
    button.setBackground(backgroundColor);
    button.setForeground(textColor);
    button.setBorderPainted(false);
    button.setFocusPainted(false);
    button.setOpaque(true);
}

/**
 * Display the products in the product catalog.
 * productPanel updated to display current list of products.
 */
private void displayProducts() {
    productPanel.removeAll();
    productPanel.setLayout(new GridLayout(0, 1, 10, 10));
```

```
        for (Product product : products) {
            JPanel itemPanel = new JPanel();
            itemPanel.setLayout(new BorderLayout(10, 10));
            itemPanel.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY, 1));

            JPanel detailsPanel = new JPanel(new GridLayout(2, 1));
            JLabel nameLabel = new JLabel(product.getName(), JLabel.CENTER);
            nameLabel.setFont(new Font("Arial", Font.BOLD, 16));
            JLabel priceLabel = new JLabel("$" + String.format("%.2f", product.getPrice()), JLabel.CENTER);
            priceLabel.setFont(new Font("Arial", Font.PLAIN, 14));

            detailsPanel.add(nameLabel);
            detailsPanel.add(priceLabel);

            JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 0));
            JButton detailsButton = new JButton("View Details");
            styleButton(detailsButton, new Color(70, 130, 180));
            detailsButton.addActionListener(e -> showProductDetails(product));

            JButton addToCartButton = new JButton("Add to Cart");
            styleButton(addToCartButton, new Color(60, 179, 113), Color.WHITE);
            styleButton(addToCartButton, new Color(60, 179, 113));
            addToCartButton.addActionListener(e -> {
                if (shoppingCart.addProduct(product)) {
                    JOptionPane.showMessageDialog(ProductView.this,
                            "Added to cart: " + product.getName());
                } else {
                    JOptionPane.showMessageDialog(ProductView.this,
                            "No more available", "Stock Alert", JOptionPane.WARNING_MESSAGE);
            }
        });

        buttonsPanel.add(detailsButton);
        buttonsPanel.add(addToCartButton);

        itemPanel.add(detailsPanel, BorderLayout.CENTER);
        itemPanel.add(buttonsPanel, BorderLayout.SOUTH);

        productPanel.add(itemPanel);
    }

    productPanel.revalidate();
    productPanel.repaint();
}

/**
 * Showing the details of a selected product in a dialog.
 * @param product is the selected product. Product can't be null.
 */
private void showProductDetails(Product product) {
    JDialog detailsDialog = new JDialog(this, "Product Details", true);
    detailsDialog.setLayout(new BorderLayout());
    detailsDialog.setSize(400, 300);
    detailsDialog.setLocationRelativeTo(this);

    JPanel detailsPanel = new JPanel();
    detailsPanel.setLayout(new BoxLayout(detailsPanel, BoxLayout.Y_AXIS));
    detailsPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    Font detailsFont = new Font("Arial", Font.PLAIN, 16);
    JLabel nameLabel = new JLabel("Name: " + product.getName());
    nameLabel.setFont(new Font("Arial", Font.BOLD, 18));
    detailsPanel.add(nameLabel);

    JLabel priceLabel = new JLabel("Price: $" + String.format("%.2f", product.getPrice()));
    priceLabel.setFont(detailsFont);
    detailsPanel.add(priceLabel);

    JLabel descriptionLabel = new JLabel("Description: " + product.getDescription());
    descriptionLabel.setFont(detailsFont);
    detailsPanel.add(descriptionLabel);

    detailsDialog.add(new JScrollPane(detailsPanel), BorderLayout.CENTER);

    JButton addToCartButton = new JButton("Add to Cart");
    styleButton(addToCartButton, new Color(60, 179, 113), Color.WHITE);
    addToCartButton.addActionListener(e -> {
        if (shoppingCart.addProduct(product)) {
            JOptionPane.showMessageDialog(detailsDialog,
                    "Added to cart: " + product.getName());
        } else {
            JOptionPane.showMessageDialog(detailsDialog,
                    "No more available", "Stock Alert", JOptionPane.WARNING_MESSAGE);
        }
    });
    JPanel buttonPanel = new JPanel();
    buttonPanel.add(addToCartButton);
    detailsDialog.add(buttonPanel, BorderLayout.SOUTH);

    detailsDialog.setVisible(true);
}
```

```
    /**
     * Displaying the contents of the shopping cart in a dialog.
     */
    private void showShoppingCart() {
        JDialog cartDialog = new JDialog(this, "Shopping Cart", true);
        cartDialog.setLayout(new BorderLayout());

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

        Font itemFont = new Font("Arial", Font.PLAIN, 16); // Modern font for cart items
        for (Map.Entry<Product, Integer> entry : shoppingCart.getProducts()) {
            Product product = entry.getKey();
            int quantity = entry.getValue();

            JPanel itemPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
            JLabel itemLabel = new JLabel(product.getName() + " - $" + product.getPrice() + " x " +
quantity);
            itemLabel.setFont(itemFont);
            itemPanel.add(itemLabel);

            JButton addButton = new JButton("+");
            styleButton(addButton, new Color(100, 149, 237), Color.WHITE);
            addButton.addActionListener(e -> {
                if (!shoppingCart.addProduct(product)) {
                    JOptionPane.showMessageDialog(cartDialog, "No more available", "Stock Alert",
JOptionPane.WARNING_MESSAGE);
                }
                cartDialog.dispose();
                showShoppingCart();
            });
            itemPanel.add(addButton);

            JButton subtractButton = new JButton("-");
            styleButton(subtractButton, new Color(255, 69, 0), Color.WHITE);
            subtractButton.addActionListener(e -> {
                shoppingCart.removeProduct(product);
                cartDialog.dispose();
                showShoppingCart();
            });
            itemPanel.add(subtractButton);

            cartPanel.add(itemPanel);
        }

        cartDialog.add(new JScrollPane(cartPanel), BorderLayout.CENTER);

        JPanel bottomPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        JLabel totalLabel = new JLabel("Total: $" + String.format("%.2f",
shoppingCart.getTotal()));
        totalLabel.setFont(new Font("Arial", Font.BOLD, 18)); // Larger and bold font for total
        totalLabel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        bottomPanel.add(totalLabel);

        JButton checkoutButton = new JButton("Checkout");
        styleButton(checkoutButton, new Color(60, 179, 113), Color.WHITE);
        checkoutButton.addActionListener(e -> {
            cartDialog.dispose();
            showCheckoutWindow();
        });
        bottomPanel.add(checkoutButton);
```

```java
        cartDialog.add(new JScrollPane(cartPanel), BorderLayout.CENTER);
        cartDialog.add(bottomPanel, BorderLayout.SOUTH);

        cartDialog.setSize(400, 300);
        cartDialog.setLocationRelativeTo(this);
        cartDialog.setVisible(true);
    }

    /**
     * Displaying the checkout window when completing the purchase.
     */
    private void showCheckoutWindow() {
        JDialog checkoutDialog = new JDialog(this, "Checkout", true);
        checkoutDialog.setLayout(new BorderLayout());
        checkoutDialog.setSize(400, 400);
        checkoutDialog.setLocationRelativeTo(this);

        JPanel summaryPanel = new JPanel();
        summaryPanel.setLayout(new BoxLayout(summaryPanel, BoxLayout.Y_AXIS));
        summaryPanel.setBorder(BorderFactory.createTitledBorder("Cart Summary"));
        Font summaryFont = new Font("Arial", Font.PLAIN, 16);
        for (Map.Entry<Product, Integer> entry : shoppingCart.getProducts()) {
            Product product = entry.getKey();
            int quantity = entry.getValue();
            JLabel productLabel = new JLabel(product.getName() + " - $" + product.getPrice() + " x " + quantity);
            productLabel.setFont(summaryFont);
            summaryPanel.add(productLabel);
        }
        checkoutDialog.add(new JScrollPane(summaryPanel), BorderLayout.NORTH);
        JPanel paymentPanel = new JPanel(new GridLayout(0, 2, 10, 10));
        paymentPanel.setBorder(BorderFactory.createTitledBorder("Payment Information"));
        paymentPanel.setOpaque(false);
        Font labelFont = new Font("Arial", Font.BOLD, 14);

        JLabel cardNumberLabel = new JLabel("Card Number:");
        cardNumberLabel.setFont(labelFont);
        JTextField cardNumberField = new JTextField();
        paymentPanel.add(cardNumberLabel);
        paymentPanel.add(cardNumberField);

        JLabel expiryDateLabel = new JLabel("Expiry Date (MM/YY):");
        expiryDateLabel.setFont(labelFont);
        JTextField expiryDateField = new JTextField();
        paymentPanel.add(expiryDateLabel);
        paymentPanel.add(expiryDateField);

        JLabel cvvLabel = new JLabel("CVV:");
        cvvLabel.setFont(labelFont);
        JTextField cvvField = new JTextField();
        paymentPanel.add(cvvLabel);
        paymentPanel.add(cvvField);

        checkoutDialog.add(paymentPanel, BorderLayout.CENTER);

        JButton confirmButton = new JButton("Confirm Purchase");
        styleButton(confirmButton, new Color(60, 179, 113), Color.WHITE);
        confirmButton.addActionListener(e -> {
            if (validatePaymentDetails(cardNumberField.getText(), expiryDateField.getText(), 
cvvField.getText())) {
```

```
                for (Map.Entry<Product, Integer> entry : shoppingCart.getProducts()) {
                    Product product = entry.getKey();
                    int quantity = entry.getValue();
                    ProductData.recordSale(product, quantity);
                    ProductData.updateProductQuantity(product, -quantity);
                }

                JOptionPane.showMessageDialog(checkoutDialog, "Purchase Successful!");
                shoppingCart.clearCart();
                checkoutDialog.dispose();
            }
        });
        checkoutDialog.add(confirmButton, BorderLayout.SOUTH);
        checkoutDialog.setSize(400, 400);
        checkoutDialog.setLocationRelativeTo(this);
        checkoutDialog.setVisible(true);
    }

    /**
     * Validation for the payment details entered by the user.
     * @param cardNumber  is the entered card number. Must be 16 digits.
     * @param expiryDate  The entered expiry date. Needs to follow MM/YY format.
     * @param cvv         The entered CVV. Must be 3 digits.
     * @return True if the payment details are valid, false otherwise.
     * cardNumber, expiryDate, and cvv can't be null.
     */
    private boolean validatePaymentDetails(String cardNumber, String expiryDate, String cvv)
{
        if (!cardNumber.matches("\\d{16}")) {
            JOptionPane.showMessageDialog(this, "Invalid Card Number. It must be 16 digits.",
"Input Error", JOptionPane.ERROR_MESSAGE);
```

```
            return false;
        }
        if (!expiryDate.matches("(0[1-9]|1[0-2])/\\d{2}")) {
            JOptionPane.showMessageDialog(this, "Invalid Expiry Date. Format must be MM/YY.", "Input Error", JOptionPane.ERROR_MESSAGE);
            return false;
        }
        if (!cvv.matches("\\d{3}")) {
            JOptionPane.showMessageDialog(this, "Invalid CVV. It must be 3 digits.", "Input Error", JOptionPane.ERROR_MESSAGE);
            return false;
        }
        return true;
    }

    /**
     * Update the quantities of purchased products in the product data.
     */
    private void updateProductQuantities() {
        for (Map.Entry<Product, Integer> entry : shoppingCart.getProducts()) {
            Product product = entry.getKey();
            int quantityPurchased = entry.getValue();
            ProductData.updateProductQuantity(product, -quantityPurchased);
        }
    }
}
```

# *SellerView.java*

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

/**
 * SellerView class that represents the seller dashboard in the application.
 */
public class SellerView extends JFrame {
    private JTabbedPane tabbedPane;
    private JPanel revenuePanel;
    private JPanel inventoryPanel;
    private JPanel addProductPanel;

    // Revenue Panel
    private JLabel totalRevenueLabel;
    private JLabel totalSalesLabel;
    private JLabel totalProfitLabel;

    // Inventory Panel
    private JTable inventoryTable;
    private DefaultTableModel inventoryModel;
    private List<Product> products;

    // Add Product Panel
    private JTextField productNameField;
    private JTextField productCostField;
    private JTextField productPriceField;
    private JTextField productQuantityField;
    private JTextField productDescriptionField;
    private JButton addProductButton;

    // Logout
    private JButton logoutButton;

    /**
     * SellerView constructor to create a window with tabs for revenue, inventory, and adding products.
     */
    public SellerView() {
        setTitle("Seller Dashboard");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        logoutButton = new JButton("Logout");
        JPanel topPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        topPanel.add(logoutButton);
        logoutButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int response = JOptionPane.showConfirmDialog(SellerView.this,
                        "Are you sure you want to log out?", "Confirm Logout",
                        JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
                if (response == JOptionPane.YES_OPTION) {
                    dispose();
                    LoginView loginView = new LoginView();
                    LoginController loginController = new LoginController(loginView);
                    loginView.setVisible(true);
                }
            }
        });

        add(topPanel, BorderLayout.NORTH);
        tabbedPane = new JTabbedPane();

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

        JPanel revenueRow = new JPanel(new FlowLayout(FlowLayout.LEFT));
        totalRevenueLabel = new JLabel("Total Revenue: $" + ProductData.getTotalRevenue());
        revenueRow.add(totalRevenueLabel);
        revenuePanel.add(revenueRow);

        JPanel salesRow = new JPanel(new FlowLayout(FlowLayout.LEFT));
        totalSalesLabel = new JLabel("Total Sales: " + ProductData.getTotalSales());
        salesRow.add(totalSalesLabel);
        revenuePanel.add(salesRow);

        JPanel profitRow = new JPanel(new FlowLayout(FlowLayout.LEFT));
        totalProfitLabel = new JLabel("Total Profit: $" + ProductData.getTotalProfit());
        profitRow.add(totalProfitLabel);
        revenuePanel.add(profitRow);

        // Inventory Panel
        initializeProducts();
        inventoryPanel = new JPanel(new BorderLayout());
```

```
        inventoryModel = new DefaultTableModel(new Object[]{"Product Name", "Cost", "Price",
"Quantity",
                                                            "Increase", "Decrease"}, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return column >= 4;
            }
        };
        inventoryTable = new JTable(inventoryModel);
        inventoryTable.getColumn("Increase").setCellRenderer(new ButtonRenderer());
        inventoryTable.getColumn("Decrease").setCellRenderer(new ButtonRenderer());
        inventoryTable.getColumn("Increase").setCellEditor(new ButtonEditor(new
JCheckBox()));
        inventoryTable.getColumn("Decrease").setCellEditor(new ButtonEditor(new
JCheckBox()));
        updateInventoryTable();
        inventoryPanel.add(new JScrollPane(inventoryTable), BorderLayout.CENTER);

        // Add Product Panel
        addProductPanel = new JPanel(new GridLayout(6, 2));
        productNameField = new JTextField(10);
        productCostField = new JTextField(10);
        productPriceField = new JTextField(10);
        productQuantityField = new JTextField(10);
        productDescriptionField = new JTextField(10);
        addProductButton = new JButton("Add Product");
        addProductButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    String name = productNameField.getText();
                    double cost = Double.parseDouble(productCostField.getText());
            double price = Double.parseDouble(productPriceField.getText());
            String description = productDescriptionField.getText();
            int quantity = Integer.parseInt(productQuantityField.getText());
            if (quantity < 0) {
                throw new IllegalArgumentException("Quantity cannot be negative.");
            }
            if (cost >= price) {
                throw new IllegalArgumentException("Cost must be less than the price.");
            }
            Product newProduct = new Product(name, cost, price, description, quantity);
            products.add(newProduct);
            ProductData.addProduct(newProduct);

            updateInventoryTable();
        } catch ( IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(SellerView.this,
                    "Invalid input: " + ex.getMessage(),
                    "Input Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});

addProductPanel.add(new JLabel("Product Name:"));
addProductPanel.add(productNameField);
addProductPanel.add(new JLabel("Cost:"));
addProductPanel.add(productCostField);
addProductPanel.add(new JLabel("Price:"));
addProductPanel.add(productPriceField);
addProductPanel.add(new JLabel("Quantity:"));
addProductPanel.add(productQuantityField);
addProductPanel.add(new JLabel("Description:"));
```

```
        addProductPanel.add(productDescriptionField);
        addProductPanel.add(addProductButton);

        tabbedPane.addTab("Revenue", revenuePanel);
        tabbedPane.addTab("Inventory", inventoryPanel);
        tabbedPane.addTab("Add Product", addProductPanel);

        add(tabbedPane);
        setLocationRelativeTo(null);
    }

    /**
     * Initializes the product list from the data source.
     * Singleton Pattern ensures that the product data is initialized only once.
     */
    private void initializeProducts() {
        products = ProductData.getProducts();
    }

    /**
     * Updates the inventory table with the current product data.
     * Observer pattern allows the inventory table to observe the changes in
     * the product data and updates itself accordingly
     */
    private void updateInventoryTable() {
        inventoryModel.setRowCount(0);
        for (Product product : products) {
            inventoryModel.addRow(new Object[]{
                product.getName(),
                product.getCost(),
                product.getPrice(),
                product.getQuantity(),
                "Increase",
                "Decrease"
            });
        }
    }

    /**
     * Refreshes the revenue data displayed on the revenue tab.
     * Observer pattern refreshes data when changes occur.
     */
    public void refreshRevenueData() {
        totalRevenueLabel.setText("Total Revenue: $" + ProductData.getTotalRevenue());
        totalSalesLabel.setText("Total Sales: " + ProductData.getTotalSales());
        totalProfitLabel.setText("Total Profit: $" + ProductData.getTotalProfit());
    }

    /**
     * Custom renderer that renders the buttons in a table cell.
     * Strategy pattern encapsulates the behavior of buttons, allowing
     * it to be easily replaced or extended.
     */
    class ButtonRenderer extends JButton implements TableCellRenderer {
        public ButtonRenderer() {
            setOpaque(true);
        }

        public Component getTableCellRendererComponent(JTable table, Object value, boolean
isSelected,
                                                       boolean hasFocus, int row, int column) {
            setText((value == null) ? "" : value.toString());
            return this;
        }
    }

    /**
     * Custom editor that handles the button clicks in a table cell.
     * Command pattern because it encapsulates the logic and information associated with button
clicks.
     */
    class ButtonEditor extends DefaultCellEditor {
        protected JButton button;
        private boolean isPushed;

        public ButtonEditor(JCheckBox checkBox) {
            super(checkBox);
            button = new JButton();
            button.setOpaque(true);
            button.addActionListener(e -> fireEditingStopped());
        }

        public Component getTableCellEditorComponent(JTable table, Object value,
                                                     boolean isSelected, int row, int column) {
            if (isSelected) {
                button.setForeground(table.getSelectionForeground());
                button.setBackground(table.getSelectionBackground());
            } else {
                button.setForeground(table.getForeground());
                button.setBackground(table.getBackground());
            }
            button.setText((value == null) ? "" : value.toString());
            isPushed = true;
            return button;
        }

        public Object getCellEditorValue() {
            if (isPushed) {
                Product product = products.get(inventoryTable.getSelectedRow());
                try {
                    if ("Increase".equals(button.getText())) {
                        product.setQuantity(product.getQuantity() + 1);
                    } else if ("Decrease".equals(button.getText())) {
                        if (product.getQuantity() <= 0) {
                            throw new IllegalArgumentException("Quantity cannot be decreased below 0.");
                        }
                        product.setQuantity(product.getQuantity() - 1);
                    }
                    updateInventoryTable();
                } catch (IllegalArgumentException ex) {
                    JOptionPane.showMessageDialog(SellerView.this,
                            ex.getMessage(),
                            "Input Error", JOptionPane.ERROR_MESSAGE);
                }
            }
            isPushed = false;
            return button.getText();
        }
    }
}
```

# *ShoppingCart.java*

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * ShoppingCart class that represents a shopping cart that holds products and its quantities.
 */
public class ShoppingCart {
    private Map<Product, Integer> products;

    public ShoppingCart() {
        products = new HashMap<>();
    }

    /**
     * Adds a product to the shopping cart.
     * @param product  is the product to be added. Product can't be null
     * @return True    if the product was added successfully, false otherwise.
     * If the product is added successfully, the quantity in the cart is increased by 1.
     */
    public boolean addProduct(Product product) {
        int currentQuantity = products.getOrDefault(product, 0);
        if (currentQuantity < product.getQuantity()) {
            products.put(product, currentQuantity + 1);
            return true;
        }
        return false;
    }

    /**
}

 * Removes a product from the shopping cart.
 * @param product is the product to be removed. can't be null.
 */
public void removeProduct(Product product) {
    int currentQuantity = products.getOrDefault(product, 0);
    if (currentQuantity > 1) {
        products.put(product, currentQuantity - 1);
    } else {
        products.remove(product);
    }
}

/**
 * Gets the set of products and quantities in the shopping cart.
 * @return the set of products and quantities.
 */
public Set<Map.Entry<Product, Integer>> getProducts() {
    return products.entrySet();
}

/**
 * Calculation of the total cost of all the products in the shopping cart.
 * @return the total cost of all the products in the shopping cart.
 */
public double getTotal() {
    double total = 0.0;
    for (Map.Entry<Product, Integer> entry : products.entrySet()) {
        total += entry.getKey().getPrice() * entry.getValue();
    }
    return total;
}
```

```
    // Clears the shopping cart by removing all the products.
    public void clearCart() {
        products.clear();
    }
}
```

# *User.java*

```java
/**
 * User class that represents the user with a username, password, and
 * user type: Customer or Seller.
 *
 */
public class User {
    private String username;
    private String password;
    private String userType;

    /**
     * User constructor that creates a new user with the
     * specified username, password, and user type.
     * @param username is the username of the user. Can't be empty or null.
     * @param password is the password of the user. Can't be empty or null.
     * @param userType is the user type: either Customer or Seller. Can't be null.
     */
    public User(String username, String password, String userType) {
        this.username = username;
        this.password = password;
        this.userType = userType;
    }

    /**
     * Gets the username of the user.
     * @return username.
     */
    public String getUsername() {
        return username;
    }
    /**
     * Gets the password of the user.
     * @return password.
     */
    public String getPassword() {
        return password;
    }

    /**
     * Gets the user type.
     * @return user type either "Customer" or "Seller".
     */
    public String getUserType() {
        return userType;
    }
}
```