Final Report

Sebastian Vega · Eduardo Paredes Guerra · Randolph Henry · Khai Lung

Functional Specifications

This application will be developed around Swing UI. The user logs in with a username and password which they authenticate. The customer and seller will have a separate login. Users will be able to view the product listings along with the product description such as name, description, price, and the quantity available. The user will be able to add products of their choosing into the shopping cart and remove them whenever they wish. Users will be able to purchase the products in the shopping cart using a credit card, assuming that the payment will always be successful. After the payment is complete, the user and seller will be able to see the order history. Users will be able to search products and filter results based on price and category. The seller will also be able to read about revenue, sales, and profit.

Platform

- Swing UI

Use cases

1. User Login
    a. Prompt customer to enter username/password
       "Please enter username and password: "
    b. System authenticates customer.
    c. System brings customer to shopping cart homepage.

2. Adding Products
    a. Seller adds new product listing by entering details (ID, price, name)
    b. System verifies inputs and adds products to the shopping cart.
3. Search and Filter Products
    a. Customer searches for product using keywords.
    b. System applies filters i.e., price range.
    c. System displays products that fit criteria.
4. Adding to Cart
    a. Customer chooses product.
    b. System adds to shopping cart.
    c. System updates shopping cart.
5. Checkout
    a. Customer begins checkout process
    b. System processes order and updates inventory.
6. View Order History

    a. Customer views history of purchases/orders.
    b. System gets and displays order history

7. View Sales Report
    a. Seller requests sales report for a specific time period.
    b. System retrieves and displays number of products sold, total revenue, and profit
8. View Profit and Loss
    a. Seller requests a profit and loss statement for a specific time period.
    b. System retrieves and displays revenues, costs, and net profit for that time period.

CRC Cards:

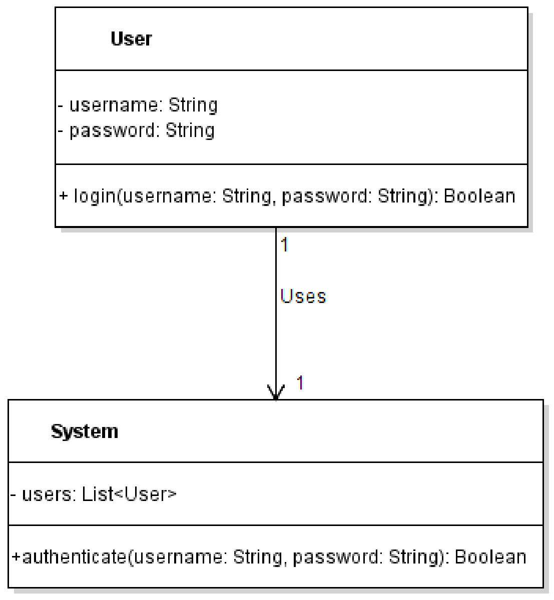
Customer:	
Responsibilities: - Login - Search and Filter products - Add products to cart - Checkout - View order history	Collaborators: - System

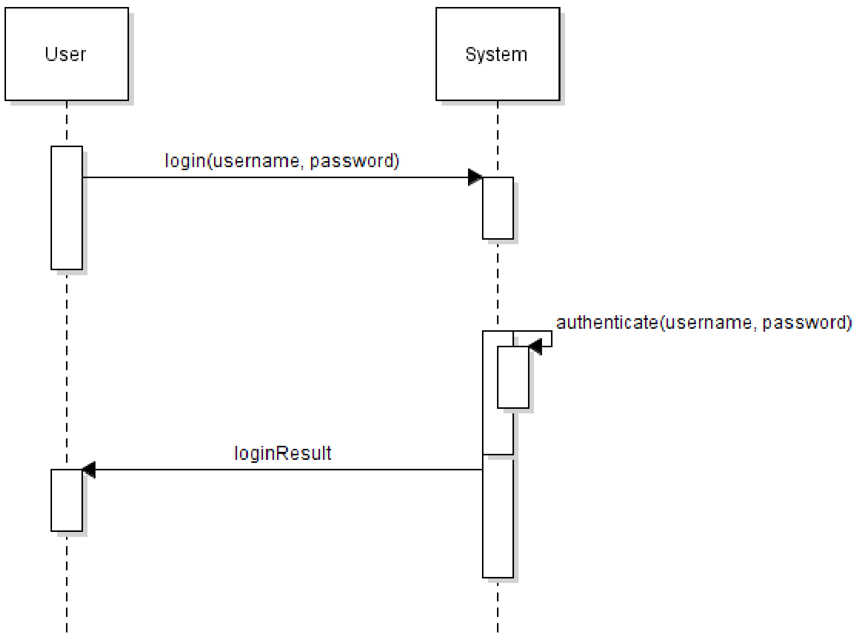
Seller:	
Responsibilities: - Login - Add products - View sales report - View profit and loss statement	Collaborators: - System

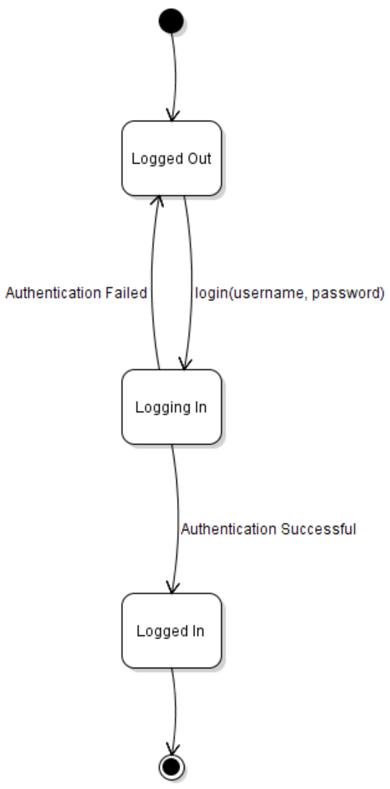
System:	
Responsibilities: - Authenticate user/seller - Verify product details - Update shopping cart - Process order and update inventory - Retrieve and display order history, sales report, profit	Collaborators: - User - Seller

UML Diagrams:

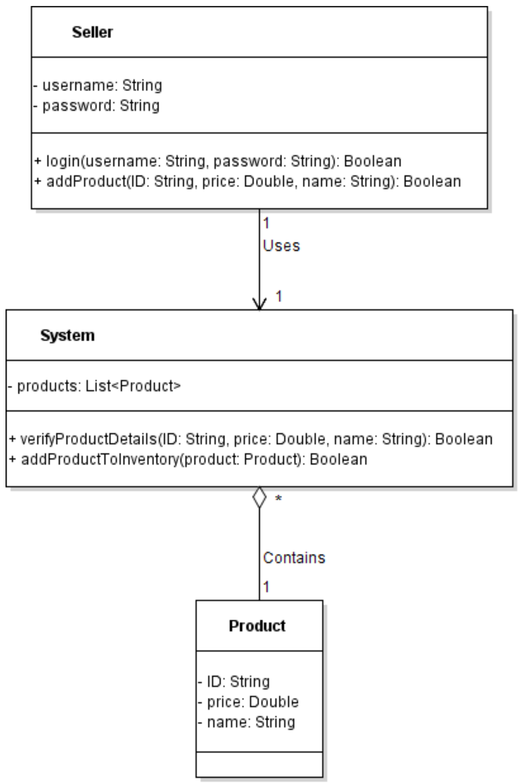
User login:

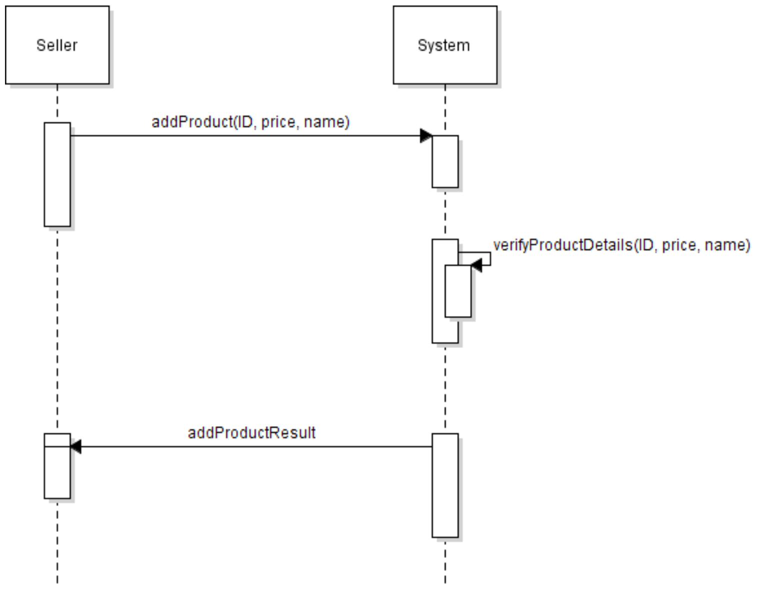

User
- username: String
- password: String
+ login(username: String, password: String): Boolean
1
Uses
1
System
- users: List<User>
+authenticate(username: String, password: String): Boolean

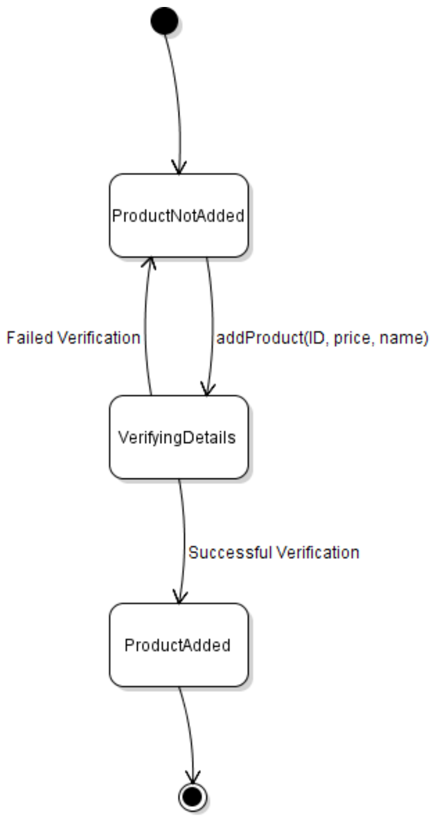




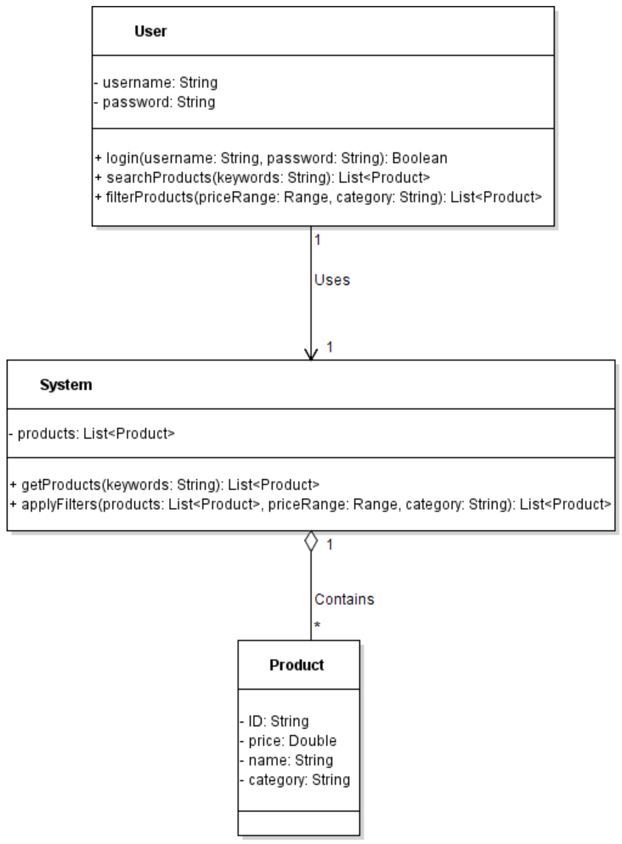
Adding Products:

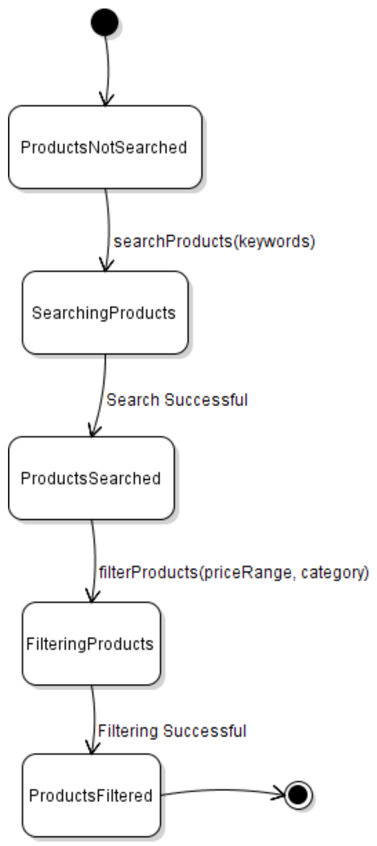

Seller
- username: String
- password: String
+ login(username: String, password: String): Boolean
+ addProduct(ID: String, price: Double, name: String): Boolean
1
Uses
1
System
- products: List<Product>
+ verifyProductDetails(ID: String, price: Double, name: String): Boolean
+ addProductToInventory(product: Product): Boolean
*
Contains
1
Product
- ID: String
- price: Double
- name: String

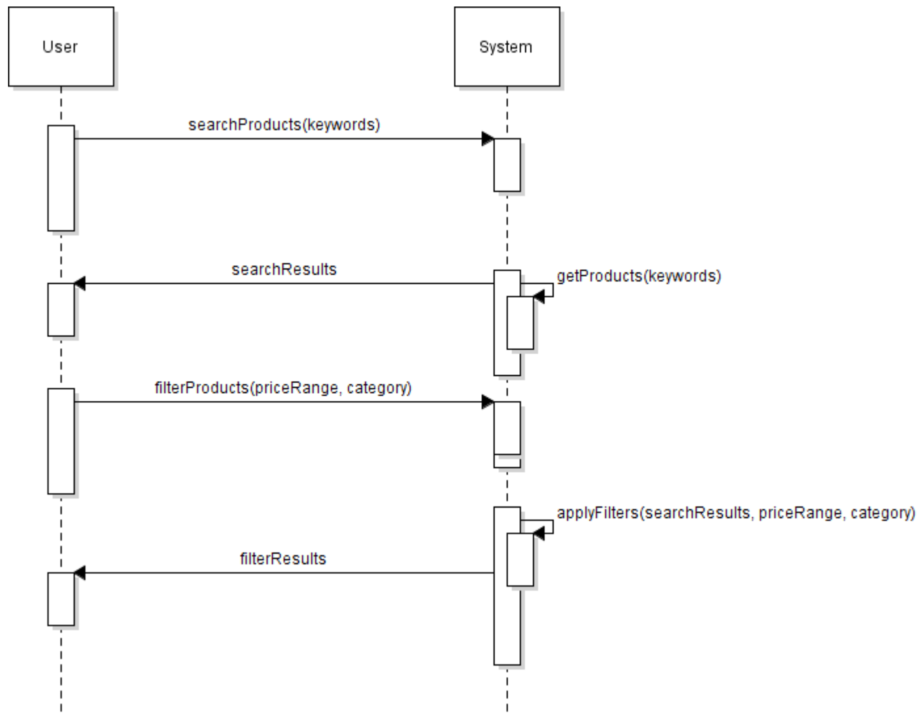

Seller
System
addProduct(ID, price, name)
verifyProductDetails(ID, price, name)
addProductResult



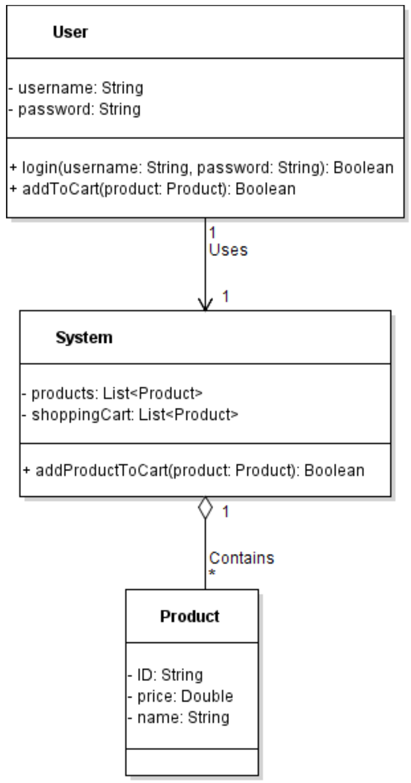
Search and Filter Products:

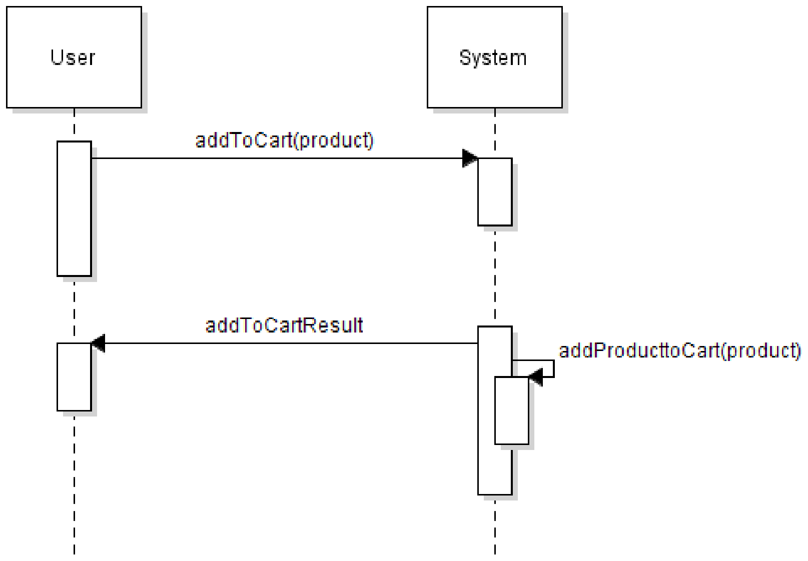

User
- username: String
- password: String
+ login(username: String, password: String): Boolean
+ searchProducts(keywords: String): List<Product>
+ filterProducts(priceRange: Range, category: String): List<Product>
1
Uses
1
System
- products: List<Product>
+ getProducts(keywords: String): List<Product>
+ applyFilters(products: List<Product>, priceRange: Range, category: String): List<Product>
1
Contains
*
Product
- ID: String
- price: Double
- name: String
- category: String

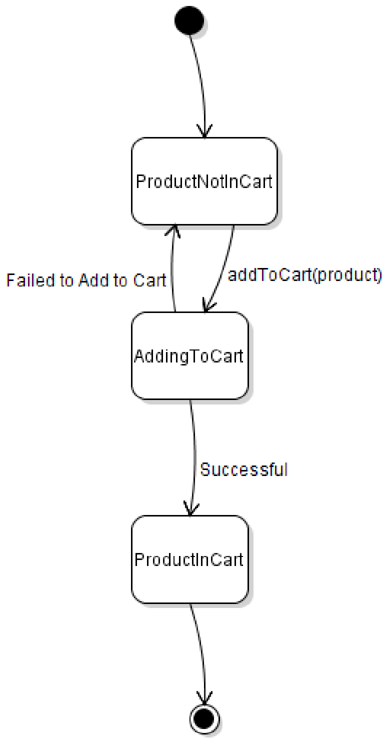



User
System
searchProducts(keywords)
searchResults
getProducts(keywords)
filterProducts(priceRange, category)
applyFilters(searchResults, priceRange, category)
filterResults

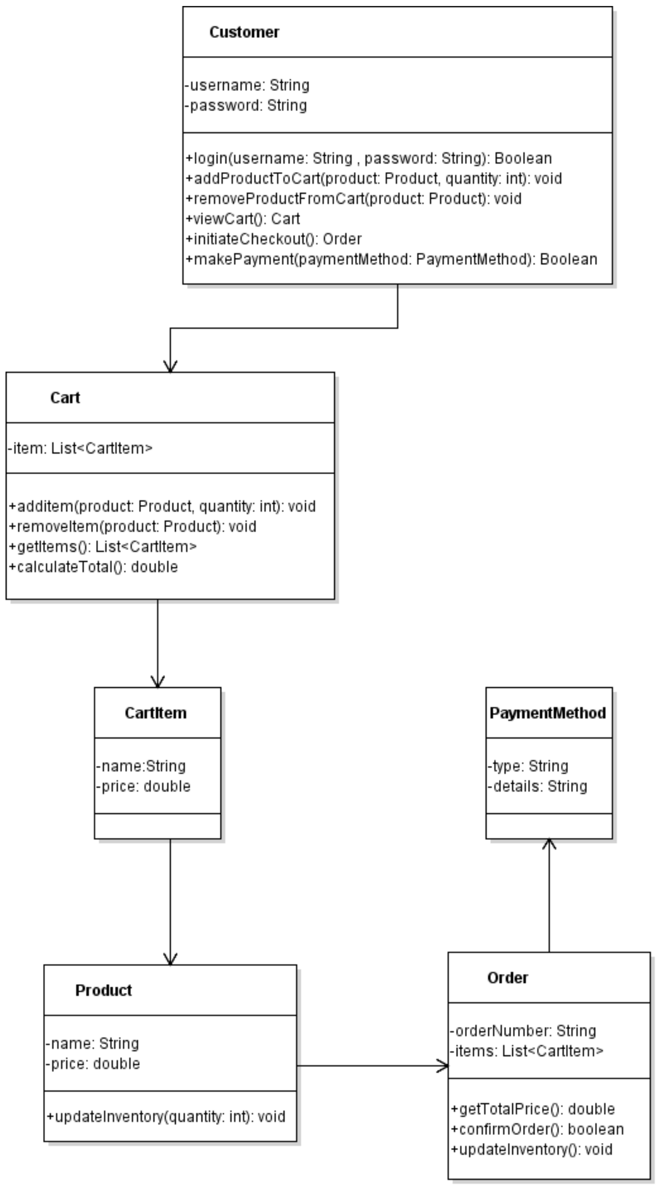
Adding to Cart:

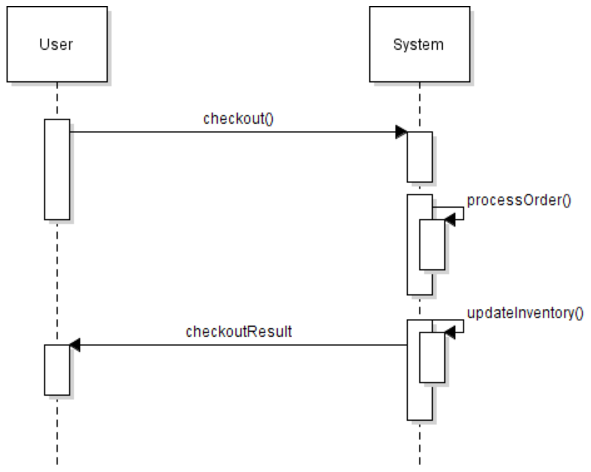

User
- username: String
- password: String
+ login(username: String, password: String): Boolean
+ addToCart(product: Product): Boolean
1
Uses
1
System
- products: List<Product>
- shoppingCart: List<Product>
+ addProductToCart(product: Product): Boolean
1
Contains
*
Product
- ID: String
- price: Double
- name: String

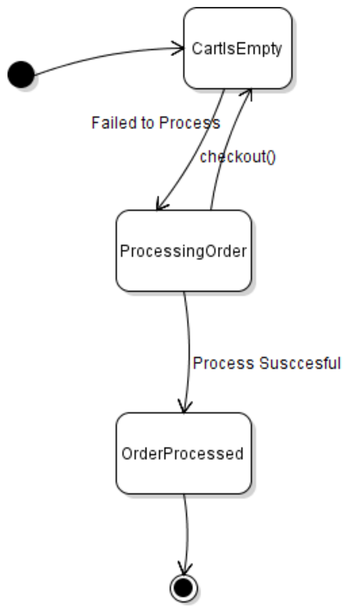




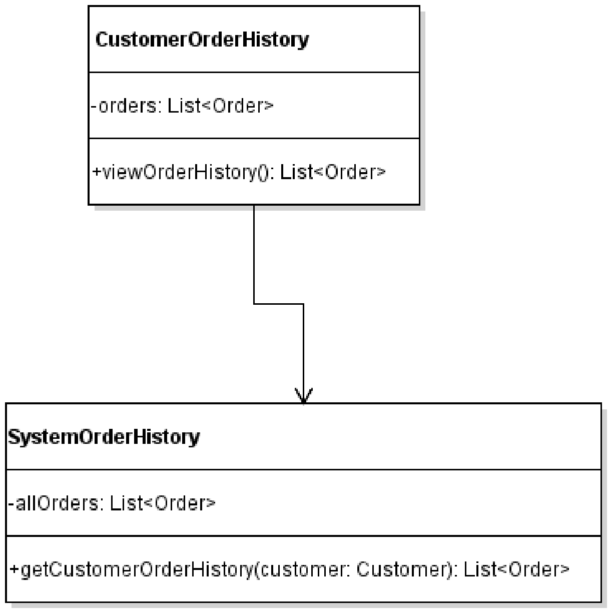
Checkout:

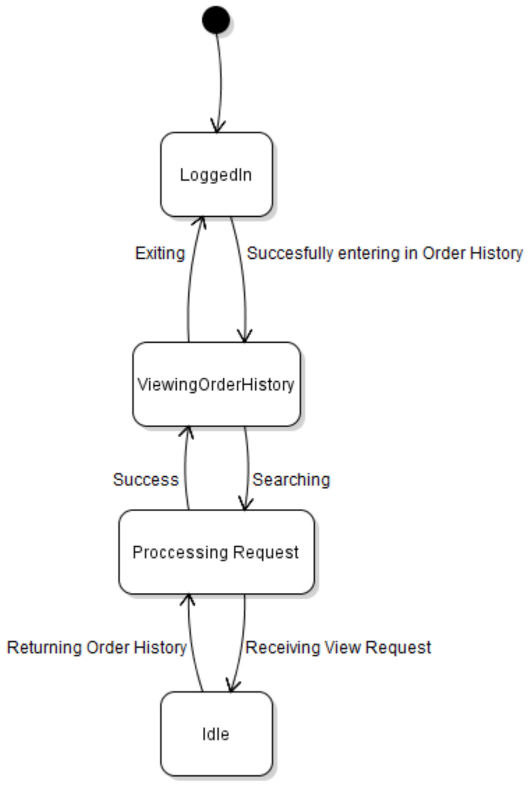

Customer
-username: String
-password: String
+login(username: String , password: String): Boolean
+addProductToCart(product: Product, quantity: int): void
+removeProductFromCart(product: Product): void
+viewCart(): Cart
+initiateCheckout(): Order
+makePayment(paymentMethod: PaymentMethod): Boolean
Cart
-item: List<CartItem>
+additem(product: Product, quantity: int): void
+removeItem(product: Product): void
+getItems(): List<CartItem>
+calculateTotal(): double
CartItem
-name:String
-price: double
PaymentMethod
-type: String
-details: String
Product
-name: String
-price: double
+updateInventory(quantity: int): void
Order
-orderNumber: String
-items: List<CartItem>
+getTotalPrice(): double
+confirmOrder(): boolean
+updateInventory(): void

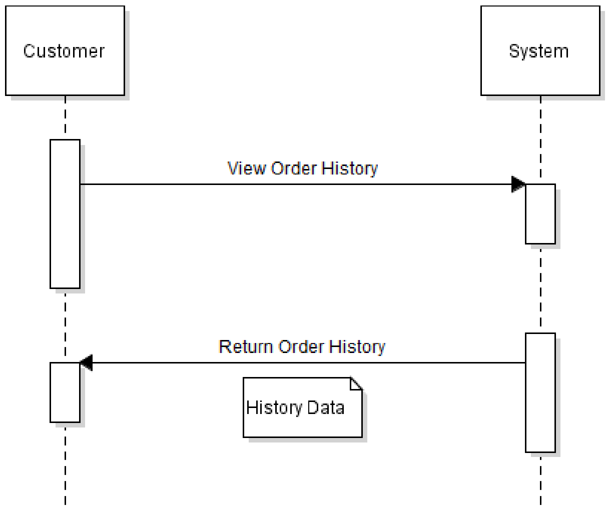

User
System
checkout()
processOrder()
updateInventory()
checkoutResult


CartIsEmpty
Failed to Process
checkout()
ProcessingOrder
Process Susccesful
OrderProcessed

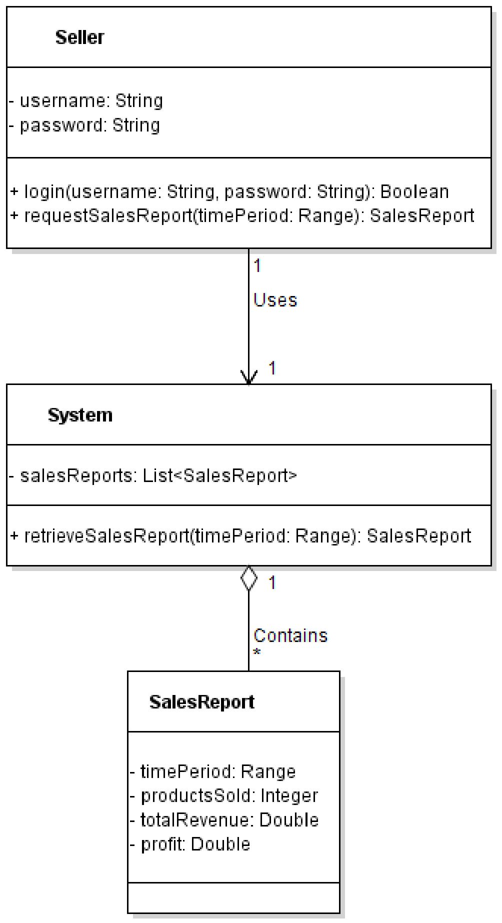
Order History:

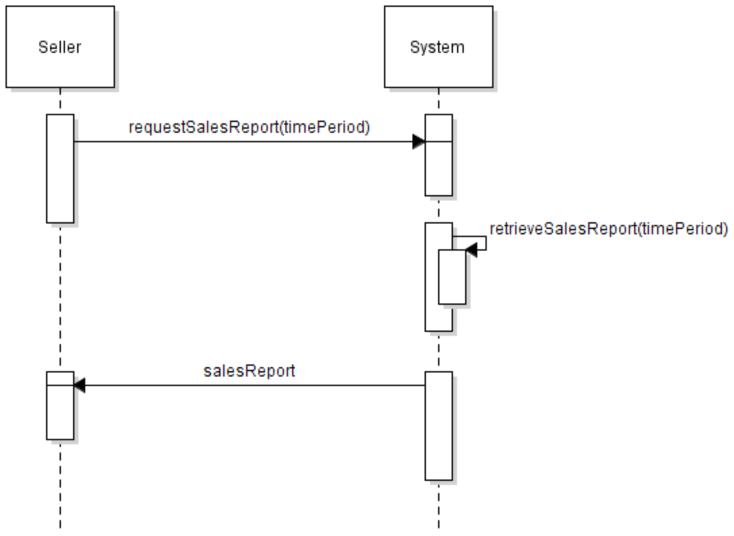

CustomerOrderHistory
-orders: List<Order>
+viewOrderHistory(): List<Order>
SystemOrderHistory
-allOrders: List<Order>
+getCustomerOrderHistory(customer: Customer): List<Order>

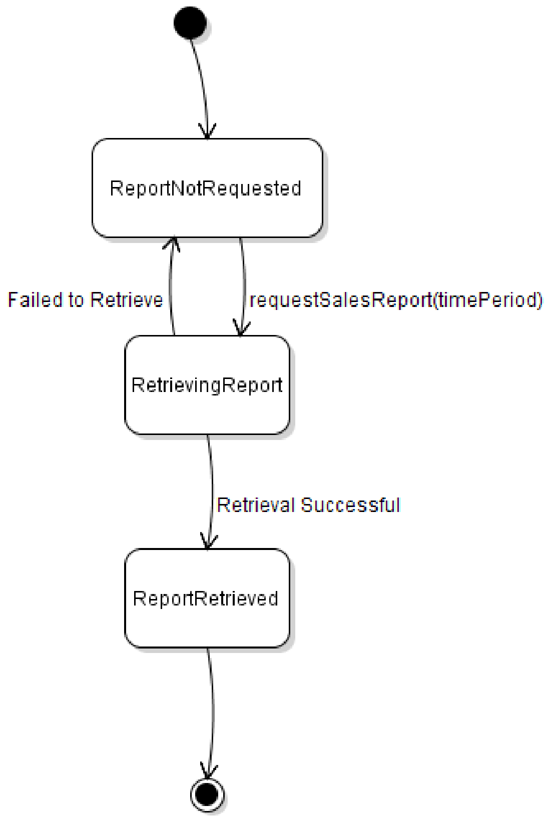




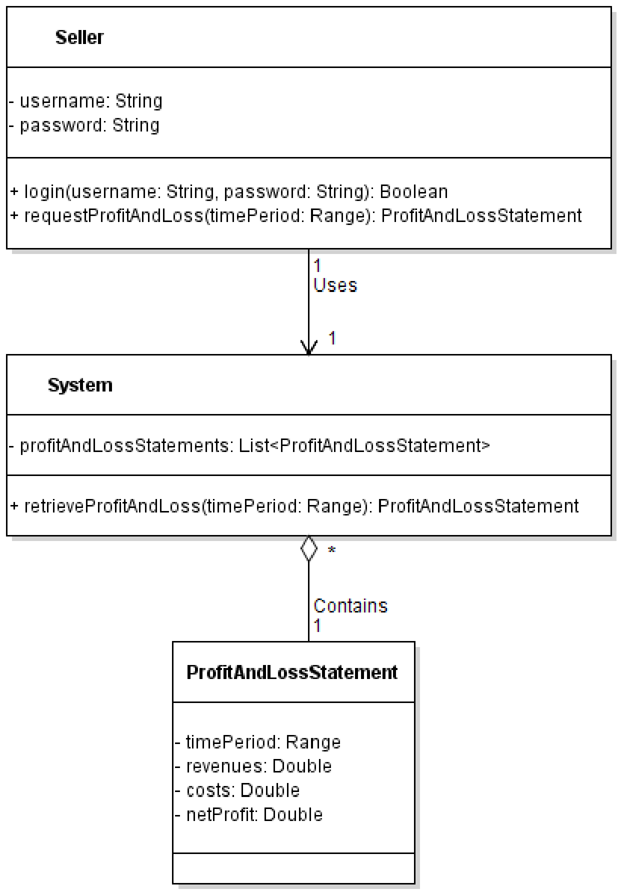
View Sales Report:

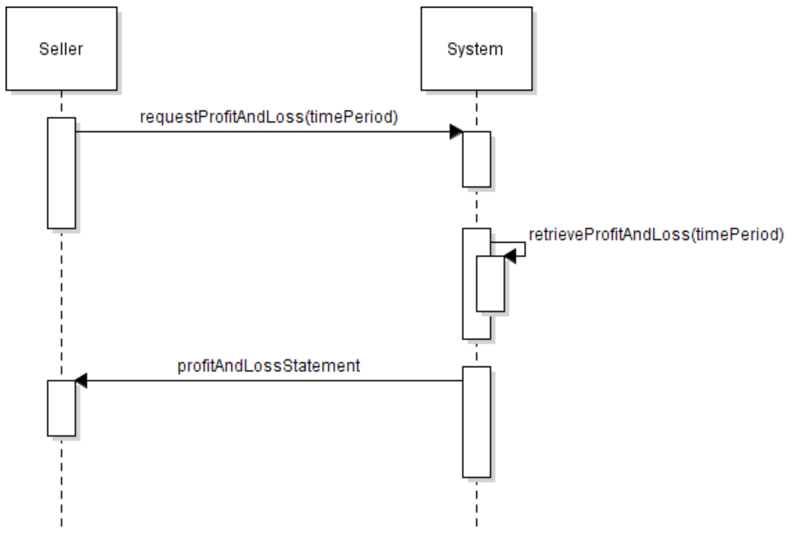

Seller
- username: String
- password: String
+ login(username: String, password: String): Boolean
+ requestSalesReport(timePeriod: Range): SalesReport
1
Uses
1
System
- salesReports: List<SalesReport>
+ retrieveSalesReport(timePeriod: Range): SalesReport
1
Contains
*
SalesReport
- timePeriod: Range
- productsSold: Integer
- totalRevenue: Double
- profit: Double

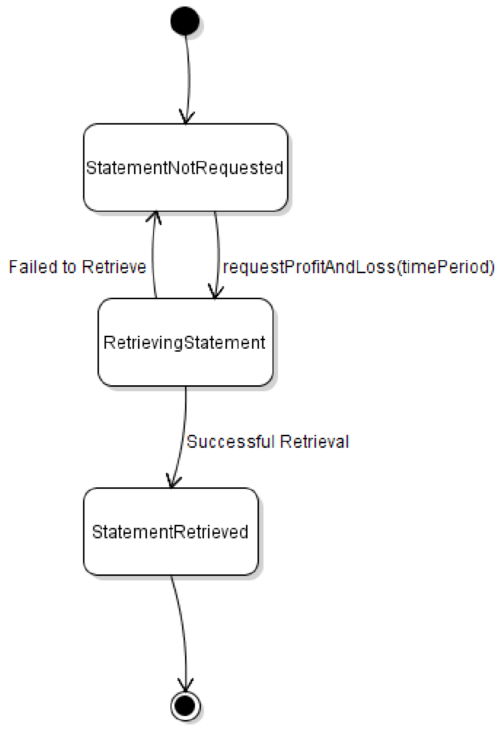

Seller
System
requestSalesReport(timePeriod)
retrieveSalesReport(timePeriod)
salesReport



View Profit and Loss:


Seller
- username: String
- password: String
+ login(username: String, password: String): Boolean
+ requestProfitAndLoss(timePeriod: Range): ProfitAndLossStatement
1
Uses
1
System
- profitAndLossStatements: List<ProfitAndLossStatement>
+ retrieveProfitAndLoss(timePeriod: Range): ProfitAndLossStatement
*
Contains
1
ProfitAndLossStatement
- timePeriod: Range
- revenues: Double
- costs: Double
- netProfit: Double


Seller
System
requestProfitAndLoss(timePeriod)
retrieveProfitAndLoss(timePeriod)
profitAndLossStatement



Source Code:

Main:

```
/**
* Main method for launching the shopping cart application
*/

public class Main {
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new LoginWindow();
            }
        });
    }
}
```

LoginWindow:

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

/**
 * Represents a login window for a shopping cart application
 */
public class LoginWindow {
    private JFrame frame;
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;

    private Map<String, Integer> failedLoginAttempts = new HashMap<>();
    private final int MAX_ATTEMPTS = 3; // Maximum failed attempts before lockout
    private final long LOCKOUT_DURATION = 30000; // Lockout duration in milliseconds
(e.g., 30000 ms = 30 seconds)
    private Map<String, Long> lockoutTime = new HashMap<>();

    /**
     * Constructor for the LoginWindow class
     * Initializes the frame and its components
     */
    public LoginWindow() {
        frame = new JFrame("eBay");
        frame.setSize(400, 250);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null); // Center the frame

        // Set the icon
        ImageIcon icon = new ImageIcon("logo.png"); // Relative path to the logo
        frame.setIconImage(icon.getImage());

        // Set a modern look and feel
        try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
        } catch (Exception e) {
            e.printStackTrace();
        }

        JPanel panel = new JPanel(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        panel.setBackground(Color.WHITE); // Set a light background
        frame.add(panel);

        placeComponents(panel, gbc);

        frame.setVisible(true);
    }

    private void placeComponents(JPanel panel, GridBagConstraints gbc) {
        gbc.insets = new Insets(4, 4, 4, 4); // Margin around components

        // Load the logo and add it as a JLabel
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 2; // Span across two columns
        gbc.anchor = GridBagConstraints.CENTER;
        ImageIcon originalIcon = new ImageIcon("logo.png");
        Image originalImage = originalIcon.getImage();
        Image resizedImage = originalImage.getScaledInstance(110, 50,
Image.SCALE_SMOOTH); // width and height are the new dimensions
        ImageIcon resizedIcon = new ImageIcon(resizedImage);
        JLabel logoLabel = new JLabel(resizedIcon);
        panel.add(logoLabel, gbc);

        // Reset gridwidth and position for the username label
        gbc.gridwidth = 1;
        gbc.gridy++;
        gbc.anchor = GridBagConstraints.WEST;
        panel.add(new JLabel("Username:"), gbc);

        gbc.gridy++;
        usernameField = new JTextField(20);
        panel.add(usernameField, gbc);

        gbc.gridy++;
        panel.add(new JLabel("Password:"), gbc);

        gbc.gridy++;
        passwordField = new JPasswordField(20);
        panel.add(passwordField, gbc);

        gbc.gridy++;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        loginButton = new JButton("Login");
        loginButton.setBackground(new Color(255, 255, 255));
        loginButton.setForeground(Color.BLACK);
        loginButton.setFocusPainted(false);
        panel.add(loginButton, gbc);

        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                handleLogin();
            }
        });
    }

    /**
     * Handles the login process when the login button is clicked.
     * This method first checks if the user is currently locked out due to too many failed
attempts.
     * If not locked out, it validates the entered username and password.
     * If validation fails, it increments the failed login attempt count and potentially locks
out the user.
     * If validation succeeds, it resets the attempt count and proceeds to the product page.
     */
    private void handleLogin() {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());

        // Check if user is currently locked out
        if (isUserLockedOut(username)) {
            JOptionPane.showMessageDialog(frame, "Account locked due to too many failed
attempts. Please try again later.");
            return;
        }

        if (isValidCustomerCredentials(username, password) ||
isValidSellerCredentials(username, password)) {
            // Successful login, reset failed attempts
            failedLoginAttempts.remove(username);
            lockoutTime.remove(username);

            // Proceed with normal login
            frame.dispose();
            new ProductPage(username.equals("c") ? "customer" : "seller");
        } else {
            // Failed login, increase attempt count
            int attempts = failedLoginAttempts.getOrDefault(username, 0);
            failedLoginAttempts.put(username, attempts + 1);

            if (attempts + 1 >= MAX_ATTEMPTS) {
                // Lock out the user
                lockoutTime.put(username, System.currentTimeMillis() +
LOCKOUT_DURATION);
                JOptionPane.showMessageDialog(frame, "Too many failed attempts. Account
locked for 30 seconds.");
            } else {
                JOptionPane.showMessageDialog(frame, "Invalid username or password.");
            }
        }
    }

    /**
     * Checks if the specified user is currently locked out.
     * A user is considered locked out if the current time is less than the stored lockout
end time.
     *
     * @param username The username to check for lockout status.
     * @return true if the user is currently locked out, false otherwise.
     */
    private boolean isUserLockedOut(String username) {
        if (!lockoutTime.containsKey(username)) {
            return false;
        }

        long lockoutEnd = lockoutTime.get(username);
        if (System.currentTimeMillis() > lockoutEnd) {
            // Lockout period has ended
            lockoutTime.remove(username);
            return false;
        }

        return true;
    }

    /**
     * Checks if the entered credentials are valid for a customer
     * @param username The entered username
     * @param password The entered password
     * @return true if the credentials are valid, false otherwise
     */
    private boolean isValidCustomerCredentials(String username, String password) {
        // Placeholder validation logic for customer
        return "c".equals(username) && "c".equals(password);
    }

    /**
     * Checks if the entered credentials are valid for a seller
     * @param username The entered username
     * @param password The entered password
     * @return true if the credentials are valid, false otherwise
     */
    private boolean isValidSellerCredentials(String username, String password) {
        // Placeholder validation logic for seller
        return "s".equals(username) && "s".equals(password);
    }
}
```

ProductPage:

```
import javax.swing.*;
import java.awt.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class ProductPage {
    private JFrame frame;
    private Cart cart; // Assuming a Cart class exists
    private Map<String, JPanel> productPanels;
    private Map<String, Double> productPrices;
    private String userRole; // Variable to store the user role

    public ProductPage(String userRole) {
        this.userRole = userRole;
        frame = new JFrame("eBay");
        frame.setSize(800, 860);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);

        // Set the icon
        ImageIcon icon = new ImageIcon("logo.png"); // Relative path to the logo
        frame.setIconImage(icon.getImage());

        cart = new Cart();
        productPanels = new HashMap<>();
        productPrices = new HashMap<>();

        JPanel mainPanel = new JPanel(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Increased
empty border
        mainPanel.setBackground(new Color(245, 245, 245));
        frame.add(mainPanel);

        // Add logo panel at the top left
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.anchor = GridBagConstraints.NORTHWEST; // Align to the top left corner
        gbc.weightx = 0; // Do not allow horizontal stretching
        gbc.insets = new Insets(-70, -50, 0, 0); // No margins
        mainPanel.add(createLogoPanel(), gbc);

        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 2;
        gbc.anchor = GridBagConstraints.CENTER;
        gbc.insets = new Insets(10, 0, 10, 0); // Increased insets for spacing between
components

        JLabel welcomeLabel = new JLabel("Welcome back, " + (userRole.equals("seller") ?
"seller!" : "customer!"));
        mainPanel.add(welcomeLabel, gbc);

        gbc.gridy++;
        setupSearchPanel(mainPanel, gbc);

        gbc.gridy++;
        loadProductsFromCSV(mainPanel, "./products.csv", gbc);

        gbc.gridy++;
        JButton viewCartButton = new JButton("View Cart");
        viewCartButton.addActionListener(e -> new CartPage(cart));
        mainPanel.add(viewCartButton, gbc);

        gbc.gridy++;
        JButton accountButton = new JButton("Account");
        accountButton.addActionListener(e -> new AccountPage(userRole));
        mainPanel.add(accountButton, gbc);

        frame.setVisible(true);
    }

    private JPanel createLogoPanel() {
        JPanel logoPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        logoPanel.setBackground(new Color(245, 245, 245)); // Match the background color
of the main panel

        ImageIcon originalIcon = new ImageIcon("logo.png");
        Image originalImage = originalIcon.getImage();
        Image resizedImage = originalImage.getScaledInstance(110, 50,
Image.SCALE_SMOOTH); // Set the desired width and height
        ImageIcon resizedIcon = new ImageIcon(resizedImage);
        JLabel logoLabel = new JLabel(resizedIcon);
        logoPanel.add(logoLabel);

        return logoPanel;
    }

    private void setupSearchPanel(JPanel panel, GridBagConstraints gbc) {
        JPanel searchPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 10)); //
Increased hgap and vgap
        searchPanel.setBackground(new Color(245, 245, 245));

        JTextField searchField = new JTextField(20);
        JTextField minPriceField = new JTextField(5);
        JTextField maxPriceField = new JTextField(5);
        JButton searchButton = new JButton("Search");

        searchPanel.add(new JLabel("Name:"));
        searchPanel.add(searchField);
        searchPanel.add(new JLabel("Min Price:"));
        searchPanel.add(minPriceField);
        searchPanel.add(new JLabel("Max Price:"));
        searchPanel.add(maxPriceField);
        searchPanel.add(searchButton);

        setupSearchButtonListener(searchField, minPriceField, maxPriceField,
searchButton);
        panel.add(searchPanel, gbc);
    }

    private void setupSearchButtonListener(JTextField searchField, JTextField
minPriceField, JTextField maxPriceField,
            JButton searchButton) {
        searchButton.addActionListener(e -> {
            String searchText = searchField.getText().toLowerCase();

            // Use 0 as default for minPrice if the field is empty
            double minPrice = minPriceField.getText().isEmpty() ? 0.0 :
parseDouble(minPriceField.getText());

            // Use a very high number as default for maxPrice if the field is empty
            double maxPrice = maxPriceField.getText().isEmpty() ? Double.MAX_VALUE
                    : parseDouble(maxPriceField.getText());

            for (Map.Entry<String, JPanel> entry : productPanels.entrySet()) {
                boolean nameMatches = entry.getKey().toLowerCase().contains(searchText);
                double price = productPrices.getOrDefault(entry.getKey(), 0.0);
                boolean priceMatches = (price >= minPrice) && (price <= maxPrice);

                entry.getValue().setVisible(nameMatches && priceMatches);
            }
        });
    }

    private void loadProductsFromCSV(JPanel panel, String filePath, GridBagConstraints
gbc) {
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            boolean firstLine = true;

            while ((line = br.readLine()) != null) {
                if (firstLine) {
                    firstLine = false; // Skip the header line
                    continue;
                }

                String[] values = line.split(",");
                if (values.length >= 2) {
                    gbc.gridy++;
                    addProductToPanel(panel, values[0], values[1], values[3], gbc);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void addProductToPanel(
            JPanel panel,
            String productName,
            String productPriceStr,
            String imageFileName,
            GridBagConstraints gbc) {
        JPanel productPanel = new JPanel(new FlowLayout());
        productPanel.setBackground(Color.WHITE); // White background for product panels
        productPanel.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY)); //
Subtle border

        // Load and display the image
        ImageIcon originalIcon = new ImageIcon(imageFileName);
        Image image = originalIcon.getImage();
        Image newimg = image.getScaledInstance(25, 25, java.awt.Image.SCALE_SMOOTH);
// Scale it to fit your layout
        ImageIcon imageIcon = new ImageIcon(newimg);
        JLabel imageLabel = new JLabel();
        imageLabel.setIcon(imageIcon);

        JLabel nameLabel = new JLabel(productName);
        JLabel priceLabel = new JLabel("$" + productPriceStr);
        JButton addButton = new JButton("Add to Cart");

        addButton.addActionListener(e -> {
            cart.addItem(productName, parseDouble(productPriceStr)); // Add item to cart
            JOptionPane.showMessageDialog(frame, productName + " added to cart!");
        });

        productPanel.add(imageLabel);
        productPanel.add(nameLabel);
        productPanel.add(priceLabel);
        productPanel.add(addButton);

        panel.add(productPanel, gbc);

        productPanels.put(productName, productPanel);
        productPrices.put(productName, Double.parseDouble(productPriceStr));
    }

    private double parseDouble(String text) {
        try {
            return Double.parseDouble(text);
        } catch (NumberFormatException e) {
            return 0.0;
        }
    }
}
```

Product:

```
/**
 * Represents a product with a name and price
 */
public class Product {
    /**
     * Name of the product
     */
    public String name;
    /**
     * Price of the product
     */
    public Double price;

    /**
     * Constructs a new Product with the specified name and price
     * @param name  the name of the product
     * @param price the price of the product
     */
    Product(String name, Double price) {
        this.price = price;
        this.name = name;
    }

    /**
     * Indicates whether some other object is "equal to" this one
     * The result is true if and only if the argument is not null and is a Product object that
represents the same name as this object
     * @param obj the reference object with which to compare
     * @return true if this object is the same as the obj argument; false otherwise
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        Product product = (Product) obj;
        return name.equals(product.name);
    }

    /**
     * Returns a hash code value for the object. This method is supported for the benefit of
hash tables such as those provided by HashMap.
     * @return a hash code value for this object
     */
    @Override
    public int hashCode() {
        return name.hashCode();
    }
}
```

AccountPage:

```java
import javax.swing.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * This class represents the account page for a user on a shopping cart application
 * The user can either be a customer or a seller
 */
public class AccountPage {
    private JFrame frame;
    private String userRole; // Customer or Seller

    /**
     * Constructs an AccountPage object and initializes the GUI
     * @param userRole the role of the user, either "customer" or "seller"
     */
    public AccountPage(String userRole) {
        this.userRole = userRole;

        frame = new JFrame("eBay Account");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setLocationRelativeTo(null); // Center the frame

        // Set the icon
        ImageIcon icon = new ImageIcon("logo.png"); // Relative path to the logo
        frame.setIconImage(icon.getImage());

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        frame.add(panel);

        if ("customer".equals(userRole)) {
            displayCustomerOrderHistory(panel);
        } else if ("seller".equals(userRole)) {
            displaySellerSalesReport(panel);
        }

        frame.setVisible(true);
    }
    /**
     * Displays the order history of a customer.
     * @param panel the JPanel to which the order history is added
     */
    private void displayCustomerOrderHistory(JPanel panel) {
        JTextArea orderHistoryArea = new JTextArea(15, 50);
        orderHistoryArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(orderHistoryArea);
        panel.add(scrollPane);

        try (BufferedReader reader = new BufferedReader(new
FileReader("customer_purchases.csv"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                orderHistoryArea.append(line + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Displays the sales report of a seller
     * @param panel the JPanel to which the sales report is added
     */
    private void displaySellerSalesReport(JPanel panel) {
        JTextArea salesReportArea = new JTextArea(15, 50);
        salesReportArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(salesReportArea);
        panel.add(scrollPane);

        try (BufferedReader reader = new BufferedReader(new
FileReader("seller_sales.csv"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                salesReportArea.append(line + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Cart:

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

/**
 * Represents a shopping cart
 */
public class Cart {
    /**
     * A map to store the products in the cart and their quantities
     */
    private Map<Product, Integer> items;

    /**
     * Constructor for the Cart class
     */
    public Cart() {
        items = new HashMap<>();
    }

    /**
     * This method adds an item to the cart
     * @param itemName The name of the item
     * @param price The price of the item
     */
    public void addItem(String itemName, Double price) {
        var item = new Product(itemName, price);
        items.put(item, items.getOrDefault(item, 0) + 1);
    }

    /**
     * This method removes an item from the cart
     * @param itemName The name of the item
     */
    public void removeItem(String itemName) {
        for (Product product : items.keySet()) {
            if (product.name.equals(itemName)) {
                int count = items.get(product);
                if (count > 1) {
                    items.put(product, count - 1);
                } else {
```

```
                items.remove(product);
            }
            break;
        }
    }
}

/**
 * This method returns a list of items in the cart
 * @return The list of products
 */
public List<Product> getItems() {
    return new ArrayList<>(items.keySet());
}

public int getItemQuantity(Product product) {
    return items.getOrDefault(product, 0);
}

/**
 * This method returns a string description of items in the cart
 * @return A string representation of items and their quantities
 */
public String getItemsDescription() {
    StringBuilder description = new StringBuilder();
    for (Map.Entry<Product, Integer> entry : items.entrySet()) {
        description.append(entry.getKey().name)
                   .append(" (Qty: ")
                   .append(entry.getValue())
                   .append("), ");
    }
    return description.length() > 0 ? description.substring(0, description.length() - 2) : "";
}

/**
 * Gets the total price of items in the cart
 * @return The total price
 */
public double getTotalAmount() {
    double total = 0;
    for (Map.Entry<Product, Integer> entry : items.entrySet()) {
        total += entry.getKey().price * entry.getValue();
    }
    return total;
    }

    /**
     * This method gets the price of a product by its name
     * @param productName The name of the product
     * @return The price of the product or null if the product is not found
     */
    public Double getProductPrice(String productName) {
        for (Product product : items.keySet()) {
            if (product.name.equals(productName)) {
                return product.price;
            }
        }
        return null; // Return null if product not found
    }
}
```

CartPage:

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

/**
 * This class represents the Cart Page in the shopping cart application
 * It displays the items in the cart, their quantity, and the total price
 * It also provides the option to remove items from the cart and proceed to
 * checkout
 */
public class CartPage {
    private JFrame frame;
    private Cart cart;
    private JButton checkoutButton; // Declare the checkout button as a class member

    /**
     * Constructor for the CartPage class
     * Initializes the frame, sets its properties, and adds components to it
     *
     * @param cart the cart object containing the items to be displayed
     */
    public CartPage(Cart cart) {
        this.cart = cart;
```

```java
        frame = new JFrame("eBay Cart");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setLocationRelativeTo(null); // Center the frame

        // Set the icon
        ImageIcon icon = new ImageIcon("logo.png"); // Relative path to the logo
        frame.setIconImage(icon.getImage());

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        frame.add(panel);

        for (Product item : cart.getItems()) {
            JPanel itemPanel = new JPanel();
            itemPanel.setLayout(new FlowLayout());

            JLabel itemLabel = new JLabel(item.name);
            JLabel priceLabel = new JLabel("$" + String.format("%.2f", item.price *
cart.getItemQuantity(item)));
            JLabel quantityLabel = new JLabel("Qty: " + cart.getItemQuantity(item));
            JButton removeButton = new JButton("Remove");

            removeButton.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    cart.removeItem(item.name);
                    frame.dispose();
                    new CartPage(cart); // Refresh the cart page
                }
            });

            itemPanel.add(itemLabel);
            itemPanel.add(priceLabel);
            itemPanel.add(quantityLabel);
            itemPanel.add(removeButton);
            panel.add(itemPanel);
        }

        // Display total price
        JLabel totalPriceLabel = new JLabel("Total Price: $" + String.format("%.2f",
cart.getTotalAmount()));
        panel.add(totalPriceLabel);
        checkoutButton = new JButton("Proceed to Checkout");
        checkoutButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                new CheckoutPage(cart); // Pass the cart object to the CheckoutPage
            }
        });

        // Disable the checkout button if the cart is empty
        checkoutButton.setEnabled(!cart.getItems().isEmpty());

        panel.add(checkoutButton);

        frame.setVisible(true);
    }
}
```

Checkout:

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

/**
 * A Checkout window for the shopping cart application.
 */
public class Checkout {
    private JFrame frame;
    private JButton checkoutButton;
    private JTextArea orderSummaryTextArea;

    /**
     * Constructor for the Checkout class
     * It initializes the GUI on the Event Dispatch Thread
     */
    public Checkout() {
        SwingUtilities.invokeLater(() -> {
            createAndShowGUI();
        });
    }

    /**
     * Sets up the GUI components and makes the frame visible
     */
```

```java
private void createAndShowGUI() {
    frame = new JFrame("Checkout");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel panel = new JPanel();
    frame.add(panel);
    placeComponents(panel);

    frame.setVisible(true);
}

/**
 * This method adds components to the panel
 * @param panel The panel to which components are added
 */
private void placeComponents(JPanel panel) {
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    checkoutButton = new JButton("Begin Checkout");
    panel.add(checkoutButton);

    orderSummaryTextArea = new JTextArea(10, 30);
    panel.add(orderSummaryTextArea);

    checkoutButton.addActionListener(new ActionListener() {
        /**
         * Defines the action to be taken when the checkout button is clicked
         * @param e The event that triggers this action
         */
        @Override
        public void actionPerformed(ActionEvent e) {
            // Add logic for processing the order and updating inventory
            orderSummaryTextArea.setText("Order Processed!\n");
        }
    });
}

/**
 * The main method that launches the Checkout window
 * @param args Command-line arguments. Not used in this application
 */
public static void main(String[] args) {
    new Checkout();
    }
}
```

CheckoutPage:

```java
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

/**
 * Represents a checkout page
 * Creates a GUI form with fields for name, email, address, and credit card information
 * Includes a "Pay Now" button to submit the form
 */
public class CheckoutPage {
    private JFrame frame;
    private Cart cart;

    /**
     * Constructor for the CheckoutPage class
     * Initializes the frame and adds all the necessary components
     */
    public CheckoutPage(Cart cart) {
        frame = new JFrame("eBay Checkout");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setLocationRelativeTo(null); // Center the frame
        this.cart = cart; // Store the cart object

        // Set the icon
        ImageIcon icon = new ImageIcon("logo.png"); // Relative path to the logo
        frame.setIconImage(icon.getImage());

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        frame.add(panel);
        // Name field
        JTextField nameField = new JTextField(20);
        panel.add(new JLabel("Name:"));
        panel.add(nameField);

        // Email field
        JTextField emailField = new JTextField(20);
        panel.add(new JLabel("Email:"));
        panel.add(emailField);

        // Address fields
        JTextField addressField = new JTextField(20);
        JTextField cityField = new JTextField(15);
        JTextField stateField = new JTextField(15);
        JTextField zipField = new JTextField(10);

        panel.add(new JLabel("Street Address:"));
        panel.add(addressField);
        panel.add(new JLabel("City:"));
        panel.add(cityField);
        panel.add(new JLabel("State:"));
        panel.add(stateField);
        panel.add(new JLabel("Zip Code:"));
        panel.add(zipField);

        // Credit Card Information
        JTextField cardNumberField = new JTextField(16);
        panel.add(new JLabel("Card Number:"));
        panel.add(cardNumberField);
        JTextField expDateField = new JTextField(16);
        panel.add(new JLabel("Exp Date:"));
        panel.add(expDateField);
        JTextField cardNumberCVVField = new JTextField(16);
        panel.add(new JLabel("CVV:"));
        panel.add(cardNumberCVVField);

        // Hard-coded shipping option
        JLabel shippingOptionLabel = new JLabel("Shipping Option: UPS Ground Delivery
(3-5 days)");
        panel.add(shippingOptionLabel);

        /**
         * Creates a "Pay Now" button and adds an action listener to it
         * When the button is clicked, it checks if all fields are filled and if the
         * card number is numeric
         * If yes, it shows a message dialog saying "Payment Processed!" and closes the
         * checkout window
         * If not, it shows a message dialog saying "Please fill all fields correctly."
         */
        JButton payNowButton = new JButton("Pay Now");
        payNowButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Perform validation here
                if (!(areAllFieldsFilled(
                        nameField,
                        emailField,
                        addressField,
                        cityField,
                        stateField,
                        zipField,
                        cardNumberField) &&
                        isNumeric(cardNumberField.getText()) &&
                        isNumeric(zipField.getText()) &&
                        isNumeric(cardNumberCVVField.getText()))) {
                    JOptionPane.showMessageDialog(frame, "Please fill all fields correctly.");
                    return;
                }

                String itemsPurchased = getItemsPurchased();
                double totalAmount = getTotalAmount();

                // Generate an order number
                String orderNumber = generateRandomOrderNumber();

                // Log customer purchase and seller sales
                logCustomerPurchase(orderNumber, emailField.getText(), itemsPurchased,
totalAmount);
                logSellerSales(orderNumber, emailField.getText(), itemsPurchased,
totalAmount);

                JOptionPane.showMessageDialog(frame, "Payment Processed!");
                frame.dispose();
            }
        });
        panel.add(payNowButton);
        frame.setVisible(true);
    }

    /**
     * Generates a random 5-digit order number
     * @return The generated order number as a string
     */
    private String generateRandomOrderNumber() {
        Random random = new Random();
        int orderNumber = random.nextInt(90000) + 10000; // Generates a random 5-digit
number
        return String.valueOf(orderNumber);
    }

    /**
     * Logs the details of a customer's purchase
     * @param orderNumber The order number of the purchase
     * @param email The email address of the customer
     * @param items The items purchased by the customer
     * @param totalAmount The total amount of the purchase.
     */
    private void logCustomerPurchase(String orderNumber, String email, String items,
double totalAmount) {
        String customerData = orderNumber + "," + email + "," + items + "," + totalAmount +
"\n";
        writeToFile("customer_purchases.csv", customerData);
    }

    /**
     * Logs the details of a seller's sales
     * @param orderNumber The order number of the sale
     * @param email The email address of the seller
     * @param items The items sold by the seller
     * @param totalAmount The total amount of the sale
     */
    private void logSellerSales(String orderNumber, String email, String items, double
totalAmount) {
        Map<String, Double> productCosts = readProductCosts();
        double totalProfit = 0.0;

        // Split the purchased items to calculate the profit for each
        String[] purchasedItems = items.split(", ");
        for (String item : purchasedItems) {
            String itemName = item.split(" \\(Qty: ")[0]; // Assuming item format is "ItemName
(Qty: X)"
            double sellingPrice = cart.getProductPrice(itemName); // Assuming Cart class has
a method to get the price
                                                     // of a product
            double costPrice = productCosts.getOrDefault(itemName, 0.0);
            double profit = sellingPrice - costPrice;
            totalProfit += profit;
        }

        String sellerData = orderNumber + "," + email + "," + items + "," + totalAmount + "," +
totalProfit + "\n";
        writeToFile("seller_sales.csv", sellerData);
    }

    /**
     * Writes data to a file
     * @param fileName The name of the file to write to
     * @param data The data to write to the file
     */
    private void writeToFile(String fileName, String data) {
        try (FileWriter fw = new FileWriter(fileName, true); BufferedWriter bw = new
BufferedWriter(fw)) {
            bw.write(data);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Gets the description of the items purchased.
     * @return A string describing the items purchased.
     */
    private String getItemsPurchased() {
        return cart.getItemsDescription(); // This method needs to be implemented in Cart
class
    }

    /**
     * Gets the total amount of the purchase
     * @return The total amount of the purchase
     */
    private double getTotalAmount() {
        return cart.getTotalAmount(); // This method needs to be implemented in Cart class
```

```java
    }

    /**
     * Reads the costs of the products from a file
     * @return A map with the product names as keys and their costs as values
     */
    private Map<String, Double> readProductCosts() {
        Map<String, Double> productCosts = new HashMap<>();
        String line;
        boolean firstLine = true; // Flag to identify the first line (header)

        try (BufferedReader br = new BufferedReader(new FileReader("products.csv"))) {
            while ((line = br.readLine()) != null) {
                if (firstLine) {
                    firstLine = false; // Skip the first line (header)
                    continue;
                }
                String[] values = line.split(",");
                if (values.length >= 3) {
                    String productName = values[0];
                    double costPrice = Double.parseDouble(values[2]); // Fetching cost price
from the third column
                    productCosts.put(productName, costPrice);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return productCosts;
    }

    /**
     * Checks if all fields in a form are filled
     * @param fields The fields to check
     * @return True if all fields are filled, false otherwise
     */
    private boolean areAllFieldsFilled(JTextField... fields) {
        for (JTextField field : fields) {
            if (field.getText().trim().isEmpty()) {
                return false;
            }
        }
        return true;
    }
    /**
     * Checks if a string can be parsed to a number
     * @param text The string to check
     * @return True if the string can be parsed to a number, false otherwise
     */
    private boolean isNumeric(String text) {
        try {
            Double.parseDouble(text);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

}
```