Final Project Report

Name: Farrell Sevillen Arya

Student ID: 2702323540

Class: L2AC

Course Name: Object Oriented Programming

Course Code: COMP6699001

Lecturer: Jude Joseph Lamug Martinez, MCS

A. Background

For the culmination of our Object-Oriented Programming course, the final project presents an opportunity to showcase the knowledge acquired throughout the semester. This project isn't just an assessment; it's a challenge to push our boundaries and apply our skills in a real-world scenario. The task at hand involves creating an innovative application that not only leverages the core concepts we've learned but also ventures beyond the classroom teachings.

In the current landscape of retail, numerous stores struggle to adapt to the digital era. The primary hurdles include complex user interfaces and the necessity for exclusive devices, making it difficult for many businesses to transition into the online realm. Recognizing this, our project aims to tackle these challenges head-on by developing an application that prioritizes simplicity, accessibility, and ease of use.

B. Project Specification

1. Introduction

There were a lot of different ideas that I wanted to do, but the one I decided to make was an e-commerce system as it was what interested me the most. I had other ideas like making a game or something else, but I couldn’t decide on what type of game it should be. The objective of the e-commerce system is to provide an easy-to-use and accessible platform for store owners to manage their inventory and for users to purchase products easily.

The user will use a device of their choice to access the application. They will open the app and be greeted with the user interface where they will have the option to browse products and add items to their shopping cart.

2. Libraries

- Java SE: Java SE is the standard library that provides core classes and APIs for Java development. It includes classes for data structures, input/output, networking, and more.
- Java Swing: Java Swing is a part of Java SE and provides a set of tools and modules for creating graphical user interfaces in desktop applications. It is a versatile library for building user interfaces.

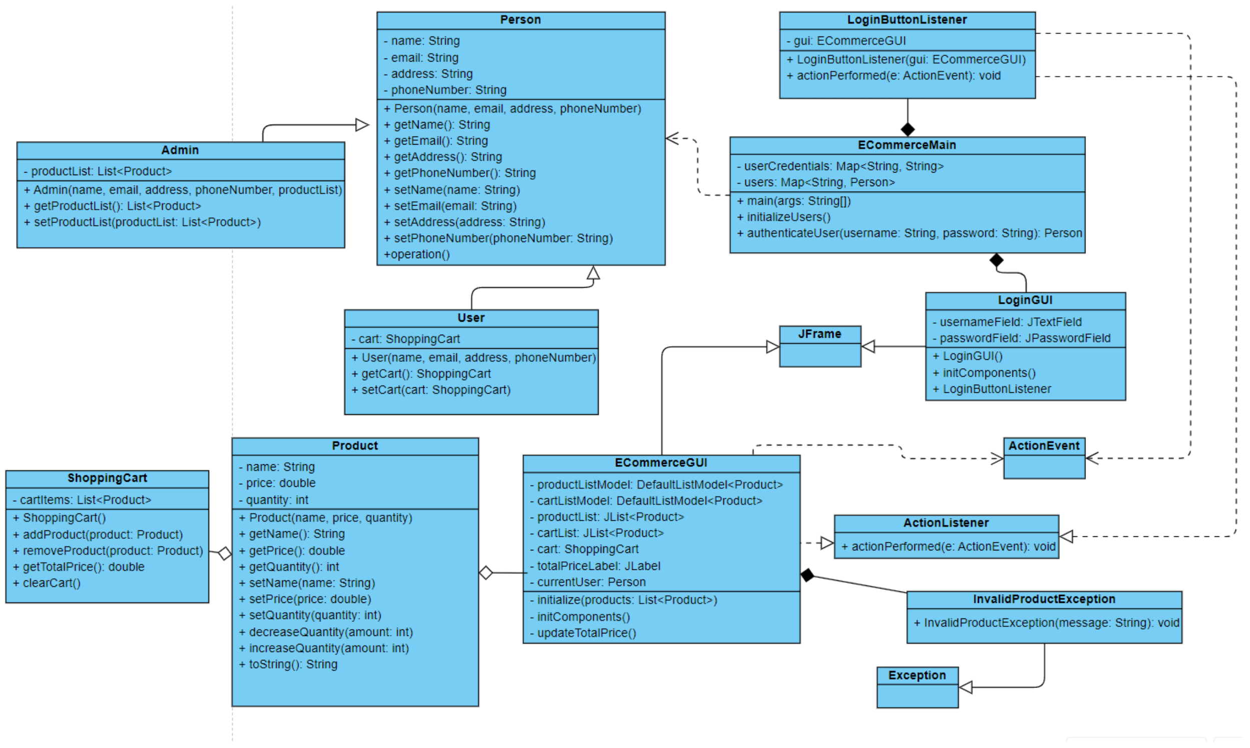
3. Important Files

- Person.java: This file includes the class which contains the code to define a person with attributes like name, email, address, and phone number.
- User.java: This file includes the class which extends Person and includes additional functionality specific to regular users, such as managing a shopping cart.
- Admin.java: This file includes the class which extends Person and includes additional functionality specific to administrators, such as managing the product inventory.
- Product.java: This file includes the class which defines a product with attributes like name, price, and quantity.
- ShoppingCart.java: This file includes the class which manages a list of products selected by the user, allowing products to be added, removed, and the total price to be calculated.

- ECommerceGUI.java: This file includes the class which provides the main graphical user interface for the application, handling user interactions and displaying the product list, shopping cart, and total price.

C. Solution Design

1. Class Diagram


Person
- name: String
- email: String
- address: String
- phoneNumber: String
+ Person(name, email, address, phoneNumber)
+ getName(): String
+ getEmail(): String
+ getAddress(): String
+ getPhoneNumber(): String
+ setName(name: String)
+ setEmail(email: String)
+ setAddress(address: String)
+ setPhoneNumber(phoneNumber: String)
+operation()
LoginButtonListener
- gui: ECommerceGUI
+ LoginButtonListener(gui: ECommerceGUI)
+ actionPerformed(e: ActionEvent): void
Admin
- productList: List<Product>
+ Admin(name, email, address, phoneNumber, productList)
+ getProductList(): List<Product>
+ setProductList(productList: List<Product>)
ECommerceMain
- userCredentials: Map<String, String>
- users: Map<String, Person>
+ main(args: String[])
+ initializeUsers()
+ authenticateUser(username: String, password: String): Person
User
- cart: ShoppingCart
+ User(name, email, address, phoneNumber)
+ getCart(): ShoppingCart
+ setCart(cart: ShoppingCart)
JFrame
LoginGUI
- usernameField: JTextField
- passwordField: JPasswordField
+ LoginGUI()
+ initComponents()
+ LoginButtonListener
ShoppingCart
- cartItems: List<Product>
+ ShoppingCart()
+ addProduct(product: Product)
+ removeProduct(product: Product)
+ getTotalPrice(): double
+ clearCart()
Product
- name: String
- price: double
- quantity: int
+ Product(name, price, quantity)
+ getName(): String
+ getPrice(): double
+ getQuantity(): int
+ setName(name: String)
+ setPrice(price: double)
+ setQuantity(quantity: int)
+ decreaseQuantity(amount: int)
+ increaseQuantity(amount: int)
+ toString(): String
ECommerceGUI
- productListModel: DefaultListModel<Product>
- cartListModel: DefaultListModel<Product>
- productList: JList<Product>
- cartList: JList<Product>
- cart: ShoppingCart
- totalPriceLabel: JLabel
- currentUser: Person
- initialize(products: List<Product>)
- initComponents()
- updateTotalPrice()
ActionEvent
ActionListener
+ actionPerformed(e: ActionEvent): void
InvalidProductException
+ InvalidProductException(message: String): void
Exception

Person:

- Base class with common attributes and methods for both `User` and `Admin`.

User:

- Inherits from `Person`.
- Has a `ShoppingCart` attribute.

Admin:

- Inherits from `Person`.

- Has a `productList` attribute which is a list of `Product`.

Product:

- Represents a product with attributes like name, price, and quantity.
- Includes methods for manipulating product data.

ShoppingCart:

- Manages a list of `Product` items.
- Includes methods for adding, removing products and calculating total price.

ECommerceGUI:

- Main GUI class.
- Has components for displaying product lists, shopping cart, and total price.
- Includes methods for user interactions such as adding/removing products and checking out.

ECommerceMain:

- Main class for starting the application.
- Handles user authentication and initializes the appropriate GUI.

LoginGUI:

- GUI for user login.
- Includes fields for username and password and login functionality.

2. How the program works

Overview

The Simple E-Commerce System program is designed to manage an inventory of products and facilitate transactions between users (customers) and the inventory. The system includes functionalities for user authentication, product management, shopping cart management, and a graphical user interface (GUI) for user interactions.

Components and Their Roles

1. **Product**: Represents an item in the inventory.
2. **Inventory**: Manages a collection of products.
3. **User**: Base class for different types of users (Admin and Customer).
4. **Admin**: Extends User, manages the inventory.

5. **Customer**: Extends User, manages a shopping cart.
6. **ECommerceGUI**: Provides a graphical user interface for interacting with the system.

Initialization

1. **Program Start**:
    - The `main` method in `ECommerceGUI` initializes and displays the GUI.
    - An initial user (Customer) is created and passed to the `ECommerceGUI` constructor.

GUI Setup

2. **GUI Initialization (`setupGUI` method in `ECommerceGUI`)**:
    - Set the title, size, and layout of the main window.
    - Create and add components for the login panel (username and password fields, login button).
    - Create and add components for displaying products and the shopping cart (text areas).
    - Create and add action buttons (add to cart, checkout, total price label).
    - Add event listeners to handle button clicks.

User Authentication

3. **Login Process (`login` method in `ECommerceGUI`)**:
    - Retrieve the entered username and password.
    - Verify that the username and password are not empty.
    - Create a new `Customer` object (assuming login is successful) and set it as the `currentUser`.
    - Display a success message and update the product display.

Inventory Management (Admin)

4. **Admin Product Management**:
    - **Adding a Product (`addProduct` method in `Admin`)**:
        - Create a new `Product` object with the given name, price, and quantity.
        - Add the product to the inventory using the `addProduct` method in `Inventory`.
    - **Updating a Product (`updateProduct` method in `Admin`)**:
        - Search for the product by name using the `searchProduct` method in `Inventory`.

        - If the product is found, update its price and quantity.
    - **Removing a Product (`removeProduct` method in `Admin`)**:
        - Remove the product from the inventory by name using the `removeProduct` method in `Inventory`.

Shopping Cart Management (Customer)

5. **Customer Shopping Cart**:
    - **Adding to Cart (`addProductToCart` method in `Customer`)**:
        - Search for the product by name using the `searchProduct` method in `Inventory`.
        - If the product is found, add it to the `cart` list.
    - **Removing from Cart (`removeProductFromCart` method in `Customer`)**:
        - Iterate through the `cart` list to find the product by name.
        - Remove the product from the list if found.
    - **Checkout (`checkout` method in `Customer`)**:
        - Initialize a `total` variable to 0.
        - Iterate through the `cart` list and add the product of the price and quantity of each product to the `total`.
        - Clear the `cart` list.
        - Return the `total`.

GUI Interaction

6. **Add to Cart (`addToCart` method in `ECommerceGUI`)**:
    - Prompt the user to enter the name of the product to add to the cart.
    - Search for the product by name using the `searchProduct` method in `Inventory`.
    - If the product is found, add it to the cart using the `addProductToCart` method in `Customer`.
    - Update the cart display.
7. **Checkout (`checkout` method in `ECommerceGUI`)**:
    - Call the `checkout` method in `Customer` to process the checkout.
    - Display the total price in a dialog box.
    - Update the total price label and the cart display.

Display Updates

8. **Update Product Display (`updateProductDisplay` method in `ECommerceGUI`)**:

    ○ Construct a string representation of all products in the inventory.
    ○ Update the `productArea` text area with this string.

9. **Update Cart Display (`updateCartDisplay` method in `ECommerceGUI`)**:
    ○ Construct a string representation of all products in the customer's cart.
    ○ Update the `cartArea` text area with this string.

Detailed Algorithm and Process Flow

D. **Initialization**:
    a. `main` method creates an instance of `ECommerceGUI`.
    b. `ECommerceGUI` constructor initializes the GUI components and sets up event listeners.

E. **Login Process**:
    a. User enters username and password.
    b. On clicking the login button, the `login` method is triggered.
    c. If credentials are valid, a new `Customer` object is created.
    d. The product display is updated to show available products.

F. **Admin Operations**:
    a. Admin logs in and gains access to product management functionalities.
    b. Admin can add, update, or remove products in the inventory.
    c. The inventory is updated accordingly, and the product display is refreshed.

G. **Customer Operations**:
    a. Customer logs in and gains access to shopping functionalities.
    b. Customer can add products to the cart by entering the product name.
    c. On clicking the add to cart button, the `addToCart` method is triggered.
    d. The cart display is updated to show the current items in the cart.

H. **Checkout Process**:
    a. Customer clicks the checkout button to proceed with the purchase.
    b. The `checkout` method calculates the total price of items in the cart.
    c. The total price is displayed, and the cart is cleared.
    d. The cart display and total price label are updated accordingly.

I. **GUI Updates**:
    a. Product and cart displays are updated dynamically based on user actions.
    b. The `updateProductDisplay` and `updateCartDisplay` methods ensure the GUI reflects the current state of the inventory and the cart.

J. Code Explanation

Product.java

Explanation

The `Product` class is a simple data class that represents an item in the e-commerce system. It encapsulates the properties of a product and provides methods to access and modify these properties.

- Attributes:
    - `name`: The name of the product.
    - `price`: The price of the product.
    - `quantity`: The number of items available in stock.
- Constructor: Initializes a new `Product` instance with the specified name, price, and quantity.
- Getters and Setters: These methods provide controlled access to the product's properties, allowing other classes to retrieve and modify the values safely.

```java
public class Product {

   private String name;

   private double price;

   private int quantity;


   public Product(String name, double price, int quantity) {

       this.name = name;

       this.price = price;

       this.quantity = quantity;

   }


   public String getName() {

       return name;

   }
```

```
    public void setName(String name) {
        this.name = name;
    }

    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;
    }
}
```

Important Parts

- **Attributes**: `name`, `price`, `quantity`
- **Getters and Setters**: `getName()`, `setName(String name)`, `getPrice()`, `setPrice(double price)`, `getQuantity()`, `setQuantity(int quantity)`

Inventory.java

Explanation

The `Inventory` class manages a collection of products. It provides methods to add, update, remove, and search for products. This class is central to managing the e-commerce system's product database.

- **Attributes**:
  - `products`: An `ArrayList` that stores `Product` instances.
- **Methods**:
  - `addProduct(Product product)`: Adds a new product to the inventory.
  - `updateProduct(String name, double price, int quantity)`: Updates the details of an existing product.
  - `removeProduct(String name)`: Removes a product from the inventory by its name.
  - `searchProduct(String name)`: Searches for a product by its name and returns the `Product` instance if found.
  - `getProducts()`: Returns the list of all products in the inventory.

```java
import java.util.ArrayList;


public class Inventory {

    private ArrayList<Product> products;


    public Inventory() {

        products = new ArrayList<>();

    }



    public void addProduct(Product product) {

        products.add(product);

    }

    public void updateProduct(String name, double price, int quantity) {
        for (Product product : products) {
            if (product.getName().equals(name)) {
                product.setPrice(price);
                product.setQuantity(quantity);
            }
        }
    }

    public void removeProduct(String name) {
        products.removeIf(product -> product.getName().equals(name));
    }

    public Product searchProduct(String name) {
        for (Product product : products) {
            if (product.getName().equals(name)) {
                return product;
            }
        }
        return null;
    }

    public ArrayList<Product> getProducts() {
        return products;
    }
}
```

Important Parts

- **Attributes**: `products`
- **Methods**: `addProduct(Product product)`, `updateProduct(String name, double price, int quantity)`, `removeProduct(String name)`, `searchProduct(String name)`, `getProducts()`

User.java

Explanation

The `User` class serves as a base class for all types of users in the system. It contains basic user information and methods for accessing and modifying this information.

- **Attributes**:
  - `username`: The username of the user.
  - `password`: The password of the user.
- **Constructor**: Initializes a new `User` instance with the specified username and password.
- **Getters and Setters**: These methods provide controlled access to the user's properties, allowing other classes to retrieve and modify the values safely.

```java
public class User {

   private String username;

   private String password;


   public User(String username, String password) {

       this.username = username;

       this.password = password;

   }


   public String getUsername() {

       return username;
```

```
    }


    public void setUsername(String username) {

        this.username = username;

    }


    public String getPassword() {

        return password;

    }


    public void setPassword(String password) {

        this.password = password;

    }

}
```

Important Parts

- **Attributes**: `username`, `password`
- **Getters and Setters**: `getUsername()`, `setUsername(String username)`, `getPassword()`, `setPassword(String password)`

Admin.java

Explanation

The `Admin` class extends the `User` class and includes additional functionalities for managing the inventory. Admins can add, update, and remove products.

- **Inheritance**: Inherits from `User`.
- **Attributes**:
    - `inventory`: An instance of the `Inventory` class.

- **Methods**:
    - `addProduct(String name, double price, int quantity)`: Adds a new product to the inventory.
    - `updateProduct(String name, double price, int quantity)`: Updates the details of an existing product.
    - `removeProduct(String name)`: Removes a product from the inventory by its name.

```java
public class Admin extends User {

   private Inventory inventory;


   public Admin(String username, String password, Inventory inventory) {

       super(username, password);

       this.inventory = inventory;

   }


   public void addProduct(String name, double price, int quantity) {

       Product product = new Product(name, price, quantity);

       inventory.addProduct(product);

   }


   public void updateProduct(String name, double price, int quantity) {

       inventory.updateProduct(name, price, quantity);

   }


   public void removeProduct(String name) {

       inventory.removeProduct(name);

   }

}
```

Important Parts

- **Inheritance**: Inherits from `User`
- **Attributes**: `inventory`
- **Methods**: `addProduct(String name, double price, int quantity)`, `updateProduct(String name, double price, int quantity)`, `removeProduct(String name)`

Customer.java

Explanation

The `Customer` class extends the `User` class and includes functionalities for shopping. Customers can add products to their cart, remove products, and checkout.

- **Inheritance**: Inherits from `User`.
- **Attributes**:
  - `cart`: An `ArrayList` that stores `Product` instances.
- **Methods**:
  - `addProductToCart(Product product)`: Adds a product to the customer's cart.
  - `removeProductFromCart(String productName)`: Removes a product from the cart by its name.
  - `checkout()`: Calculates the total price of all items in the cart and clears the cart.
  - `getCart()`: Returns the list of products in the cart.

```java
import java.util.ArrayList;


public class Customer extends User {

   private ArrayList<Product> cart;


   public Customer(String username, String password) {

       super(username, password);

       cart = new ArrayList<>();
```

```
    }


    public void addProductToCart(Product product) {
        cart.add(product);
    }


    public void removeProductFromCart(String productName) {
        cart.removeIf(product -> product.getName().equals(productName));
    }


    public double checkout() {
        double total = 0;
        for (Product product : cart) {
            total += product.getPrice() * product.getQuantity();
        }
        cart.clear();
        return total;
    }


    public ArrayList<Product> getCart() {
        return cart;
    }
}
```

Important Parts

- **Inheritance**: Inherits from `User`

- **Attributes**: `cart`
- **Methods**: `addProductToCart(Product product)`, `removeProductFromCart(String productName)`, `checkout()`, `getCart()`

EccomerceGUI.java

Explanation

The `ECommerceGUI` class manages the graphical user interface. It handles user interactions, such as logging in, adding products to the cart, and checking out. It uses Java Swing components to create the interface.

- **Attributes**:
    - Various Swing components (`JTextArea`, `JTextField`, `JButton`, etc.)
    - `inventory`: An instance of the `Inventory` class.
    - `currentUser`: The currently logged-in user.
- **Methods**:
    - `setupGUI()`: Initializes and sets up the GUI components.
    - `login()`: Handles user login.
    - `addToCart()`: Adds selected products to the cart.
    - `checkout()`: Handles the checkout process.
    - `updateProductDisplay()`: Updates the display of products.
    - `updateCartDisplay()`: Updates the display of the cart.

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


public class ECommerceGUI extends JFrame {

   private Inventory inventory;

   private User currentUser;

   private JTextArea productArea;

   private JTextArea cartArea;
   private JTextField usernameField;

   private JPasswordField passwordField;

   private JButton loginButton;

   private JButton addToCartButton;

   private JButton checkoutButton;

   private JLabel totalPriceLabel;


   public ECommerceGUI(User user) {

       this.currentUser = user;

       this.inventory = new Inventory();

       setupGUI();

   }


   private void setupGUI() {

       // Initialize and set up the GUI components

       setTitle("E-Commerce System");

       setSize(600, 400);

       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       setLayout(new BorderLayout());


       // Login panel

       JPanel loginPanel = new JPanel(new GridLayout(3, 2));

       loginPanel.add(new JLabel("Username:"));

       usernameField = new JTextField();

       loginPanel.add(usernameField);

       loginPanel.add(new JLabel("Password:"));
```

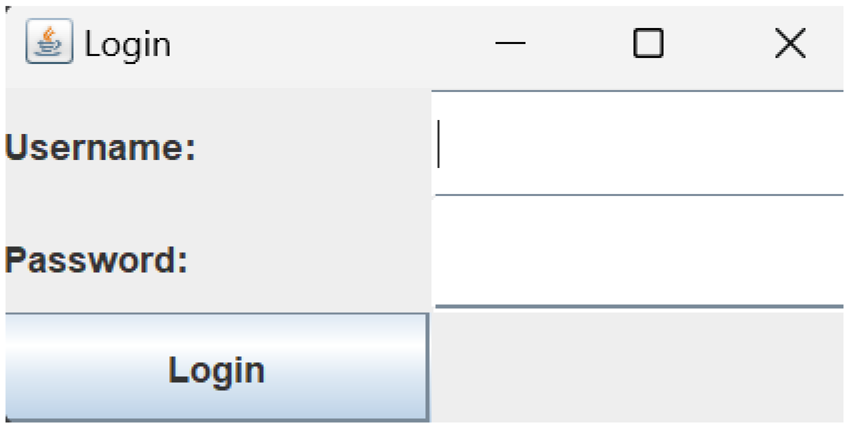
```
        passwordField = new JPasswordField();

        loginPanel.add(passwordField);

        loginButton = new JButton("Login");

        loginPanel.add(loginButton);

        add(loginPanel, BorderLayout.NORTH);


        // Product display area

        productArea = new JTextArea();

        add(new JScrollPane(productArea), BorderLayout.CENTER);


        // Cart display area

        cartArea = new JTextArea();

        add(new JScrollPane(cartArea), BorderLayout.EAST);


        // Action buttons

        JPanel actionPanel = new JPanel(new GridLayout(1, 3));

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

        actionPanel.add(addToCartButton);

        checkoutButton = new JButton("Checkout");

        actionPanel.add(checkoutButton);

        totalPriceLabel = new JLabel("Total: $0.0");

        actionPanel.add(totalPriceLabel);

        add(actionPanel, BorderLayout.SOUTH);


        // Event listeners

        loginButton.addActionListener(e -> login());
        addToCartButton.addActionListener(e -> addToCart());

        checkoutButton.addActionListener(e -> checkout());

    }


    private void login() {

        // Handle user login

        String username = usernameField.getText();

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

        // For simplicity, just check if username and password are not empty

        if (!username.isEmpty() && !password.isEmpty()) {

            currentUser = new Customer(username, password); // Assume login is
successful and user is a customer

            JOptionPane.showMessageDialog(this, "Login successful!");

            updateProductDisplay();

        } else {

            JOptionPane.showMessageDialog(this, "Login failed! Please enter
username and password.");

        }

    }


    private void addToCart() {

        // Handle adding products to the cart

        String selectedProduct = JOptionPane.showInputDialog(this, "Enter
product name to add to cart:");

        Product product = inventory.searchProduct(selectedProduct);

        if (product != null) {

            if (currentUser instanceof Customer) {
                ((Customer) currentUser).addProductToCart(product);

                updateCartDisplay();

            }

        } else {

            JOptionPane.showMessageDialog(this, "Product not found!");

        }

    }



    private void checkout() {

        // Handle the checkout process

        if (currentUser instanceof Customer) {

            double total = ((Customer) currentUser).checkout();

            JOptionPane.showMessageDialog(this, "Checkout successful! Total: $"
+ total);

            totalPriceLabel.setText("Total: $" + total);

            updateCartDisplay();

        }

    }



    private void updateProductDisplay() {

        // Update the display of products

        StringBuilder productDisplay = new StringBuilder("Available
Products:\n");

        for (Product product : inventory.getProducts()) {

            productDisplay.append(product.getName()).append(" -
$").append(product.getPrice()).append("
(").append(product.getQuantity()).append(")\n");

        }
```

```java
        productArea.setText(productDisplay.toString());

    }



    private void updateCartDisplay() {

        // Update the display of the cart

        if (currentUser instanceof Customer) {

            StringBuilder cartDisplay = new StringBuilder("Shopping Cart:\n");

            for (Product product : ((Customer) currentUser).getCart()) {

                cartDisplay.append(product.getName()).append(" -
$").append(product.getPrice()).append("
(").append(product.getQuantity()).append(")\n");

            }

            cartArea.setText(cartDisplay.toString());

        }

    }



    public static void main(String[] args) {

        // Create and display the GUI

        SwingUtilities.invokeLater(() -> {

            ECommerceGUI gui = new ECommerceGUI(new Customer("user1",
"password"));

            gui.setVisible(true);

        });

    }

}
```

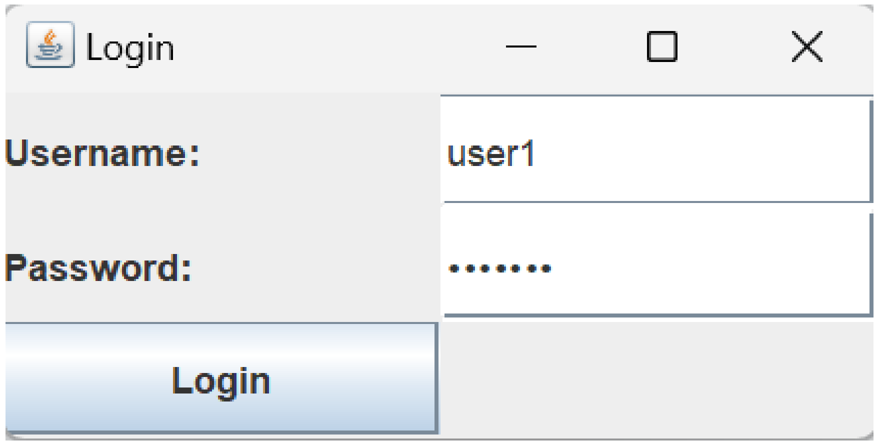
Important Parts

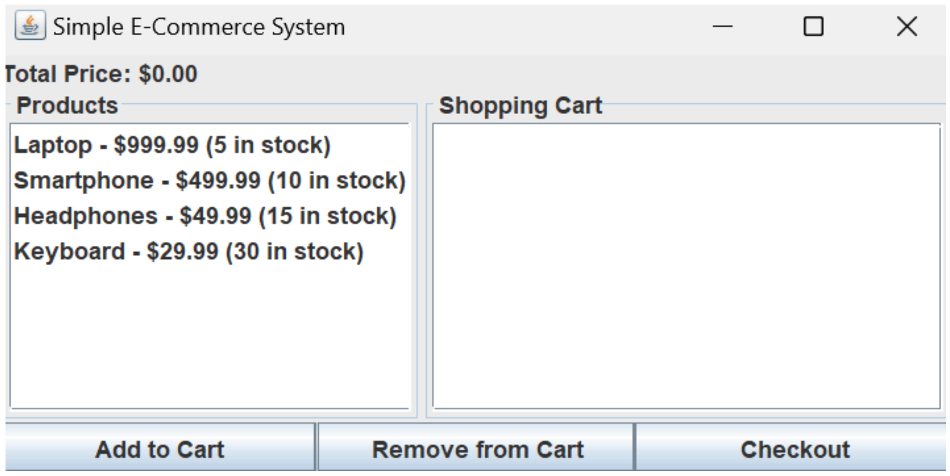
- **Attributes**: Swing components like `productArea`, `cartArea`, `loginButton`, `addToCartButton`, `checkoutButton`, `totalPriceLabel`
- **Methods**: `setupGUI()`, `login()`, `addToCart()`, `checkout()`, `updateProductDisplay()`, `updateCartDisplay()`

K. Application Evidence



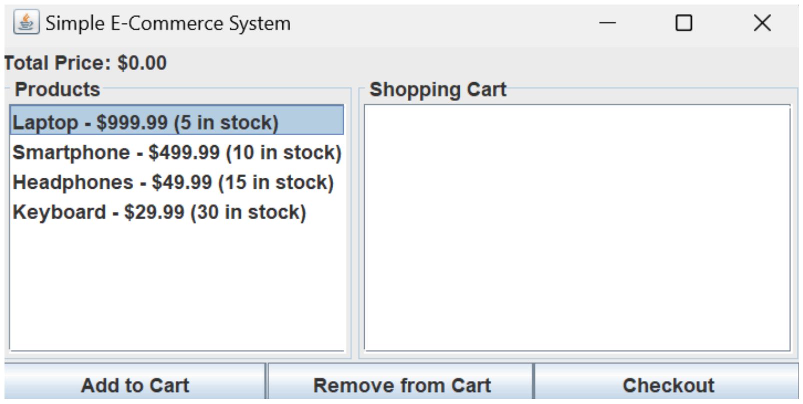
This is the app when it is booted up





Laptop - $999.99 (5 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Successfully logging in

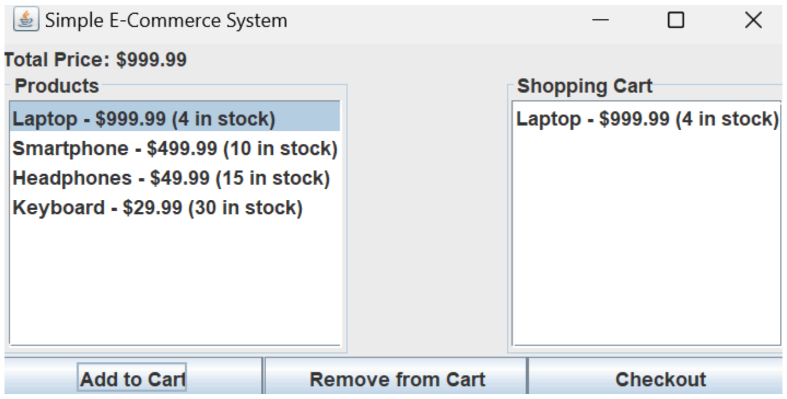


Total Price: $0.00

Products

Laptop - $999.99 (5 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Shopping Cart



Total Price: $999.99

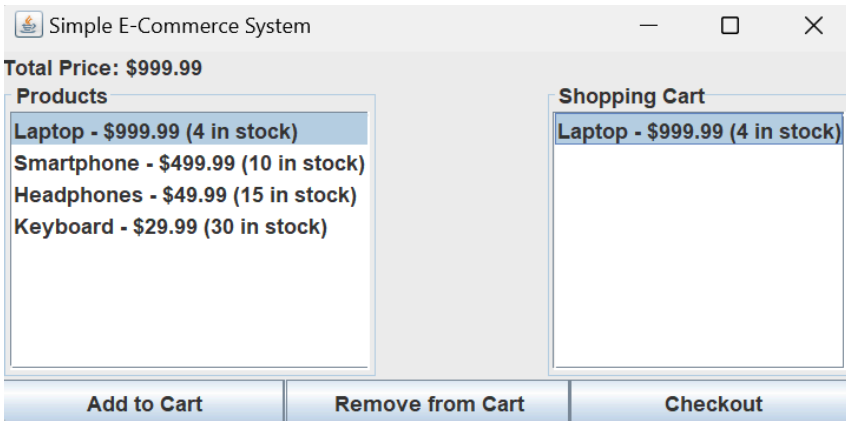
Products

Laptop - $999.99 (4 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Shopping Cart

Laptop - $999.99 (4 in stock)

Adding an item


Simple E-Commerce System

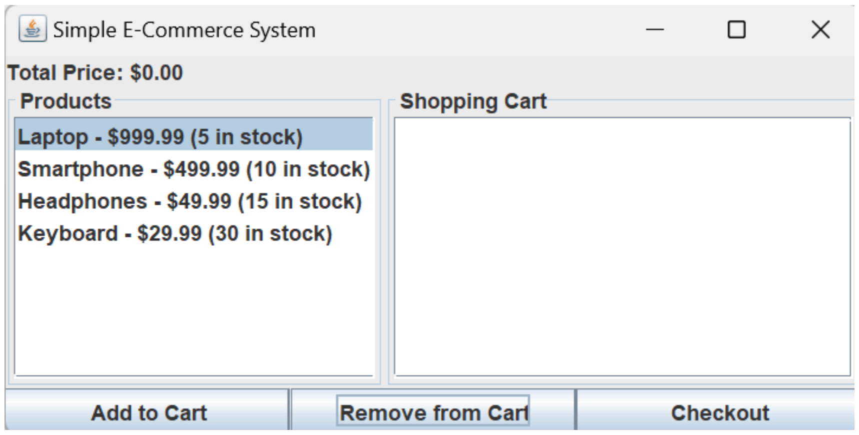
Total Price: $999.99

Products

Laptop - $999.99 (4 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Shopping Cart

Laptop - $999.99 (4 in stock)


Simple E-Commerce System

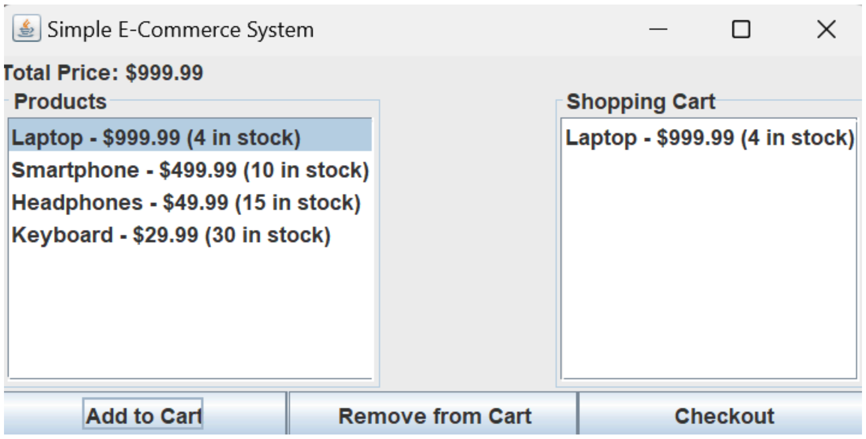
Total Price: $0.00

Products

Laptop - $999.99 (5 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Shopping Cart

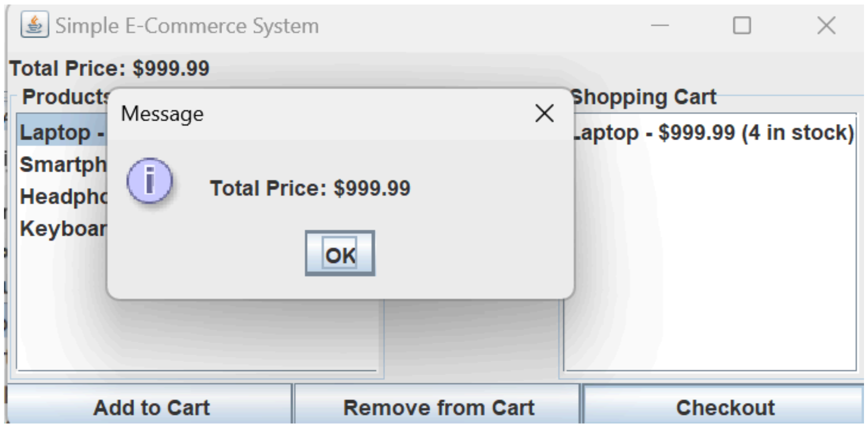
Removing an item



Total Price: $999.99

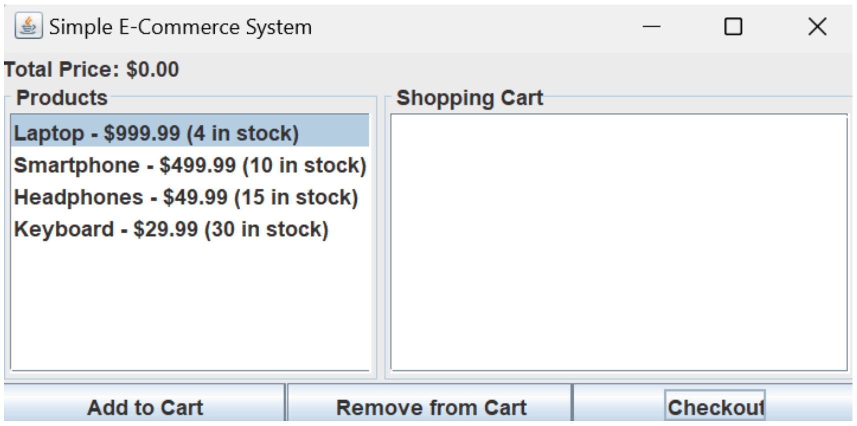
Laptop - $999.99 (4 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Laptop - $999.99 (4 in stock)



Total Price: $999.99

Total Price: $999.99



Total Price: $0.00

Laptop - $999.99 (4 in stock)
Smartphone - $499.99 (10 in stock)
Headphones - $49.99 (15 in stock)
Keyboard - $29.99 (30 in stock)

Successful checkout

L. GithubLink: https://github.com/RelKereta/OOP-final

References:

- https://dev.to/
- https://codebun.com/
- https://www.javaguides.net/2021/02/java-free-e-commerce-open-source-projects.html
- https://data-flair.training/blogs/online-shopping-system-java-jsp-servlets/