Shopping Cart Project Final Report
Group 19 – John Garofalo, Rafael Gutierrez,
Tivvon Cruickshank, Valente Lowery
COP 4331-002
12/7/2023

Shopping Cart using SwingUI

Table of Contents

Glossary (Rafael)

Term	Definition
Application UI	User interface of the application
Availability	The status on if a product is in stock
Browse Window	User interface screen that shows products
Checkout	Finalizing a purchase
Inventory	List of products a seller has available
Invoice Price	Pirce that a seller purchases a product from a supplier
Item ID	Unique identifier for a product
Log In	The process of entering user information to access an account
Password	Combination of characters used to verify a user
Product	Item available for sale
Profits	Revenues minus cost
Quantity	Number of units available for a product
Revenues	Sum of sell prices for all sold items
Sell Price	Price that the seller sets for customers to purchase
Seller	Any individual that sells products
Seller Inventory Page	User interface where sellers can manage their inventory
Shipping Information	A customer's shipping method, payment method, and address
Shopping Cart	Virtual page where users can see selected products they plan to buy
Transaction	A purchase or sale involving the exchange of money
Username	Unique identifier used to access a user's account

Functional Specification (John)

A customer logs in with their username and password and the window loads a menu where they can browse through a list of available items that includes the product name, price, and available quantity. From this window the user can select products and add them to their shopping cart, or they can click on a product to get the full description including pricing and availability in a pop-up window. If the product is available, the customer will be able to add it to their shopping cart. The total number of products currently in the shopping cart is kept on the main product browse window.

The customer can proceed to checkout at any time. On the shopping cart window, the cart can be updated by changing the quantity of each product in the cart, then the user can confirm their cart and checkout. At checkout, the customer verifies the contents of their cart and pays by entering credit card information.

The product browse window contains an option for the user to switch to their seller inventory page. On this page, the user (now as a seller) can update their inventory by adding products, which consists of specifying the product's name, invoice price, sell price and updating the available quantity.

The internal product representation includes item ID, type, quantity, invoice price and selling price. The application keeps track of all costs (sum of invoice price for all items bought in inventory), revenues (sum of sell price for all sold items) and profits (revenues minus costs). The seller can access this information from the application UI. The platform used for implementation is SwingUI.

Essential Use Cases (John)

1. User Logs In
2. Customer Adds Items to Shopping Cart
3. Customer Reviews Product Details
4. Customer Reviews/Updates Shopping Cart
5. Customer Checks Out
6. Seller Reviews/Updates Inventory
7. Seller Reviews Profits
8. Seller Adds New Product

Detailed Use Cases

User Logs In: (John)

1. User opens the application.
2. Application prompts the user to enter their username.
3. User enters their username.
4. Application prompts the user to enter their password.
5. User enters their password.
6. User is recognized as a valid customer by the application.
7. Application loads the product browse window.

Customer Adds Items to Shopping Cart: (John)

1. Customer carries out *Log In*.
2. Customer scrolls through the list of available items.
3. Customer finds an item of interest and clicks the “add to cart” button next to the item’s description (name, price, availability).
4. Item is available and is therefore added to the customer’s cart.
5. Value representing the number of items in the customer’s cart as displayed in the product browse window is incremented by one.
6. Continue with Step 2.

Customer Reviews Product Details: (John)

1. Customer carries out *Log In*.
2. Customer scrolls through the list of available items.
3. Customer finds an item of interest and clicks on it.
4. Pop-up window opens with the product's full details including name, price, availability, and an expanded description.
5. Customer looks over the product information.
6. Customer clicks the "add to cart" button.
7. Item is available and is therefore added to the customer's cart.
8. Value representing the number of items in the customer's cart as displayed in the product browse window is incremented by one.
9. Customer closes the product details window.
10. Customer returns to the product browse window.

Customer Reviews/Updates Shopping Cart: (John)

1. Customer carries out *Customer Adds Items to Shopping Cart*.
2. Customer clicks Shopping Cart button.
3. Pop-up window opens with the details of all products in the cart including name, price, availability and quantity in the cart.
4. Customer looks over the product information.
5. Customer makes any changes to the items in their cart including deleting items from their cart or changing the quantity of different products.
6. Customer clicks on finalize button to confirm changes to their shopping cart.
7. Number of items in the cart is updated in the product browse window.

Customer Checks Out: (John)

1. Customer carries out *Customer Reviews/Updates Shopping Cart*.
2. Customer clicks Check Out button to confirm shopping cart.
3. Customer is prompted to enter their credit card information.
4. Customer enters their credit card information.
5. Customer is prompted to enter their shipping information.
6. Customer enters their shipping information.
7. Shopping cart items, total price, credit card info, and shipping info are displayed to the customer for confirmation.
8. Customer selects Confirm button to finalize order.
9. Credit card and shipping information are saved in the customer's account under billing information.
10. Receipt is sent to the customer account as order confirmation.

Seller Reviews/Updates Inventory: (John)

1. Seller carries out *Log In*.
2. Seller clicks the button to look over/update inventory.
3. Application loads the seller's product inventory.
4. Seller scrolls through their inventory.
5. Seller clicks an item to review/update.
6. Seller changes item details such as the price and available quantity.
7. Seller confirms the changes.
8. Item is updated in the seller's inventory.

Seller Reviews Profits: (John, Rafael)

1. Seller carries out *Log In.*
2. Seller clicks the button to review monetary statistics.
3. Application opens a pop-up window displaying current totals for costs, revenue, and profits, as well as in-depth information for each individual product in each group.
4. Seller scrolls through their statistics page.
5. Seller closes the pop-up window.

Seller Adds New Product: (John, Rafael)

1. Seller carries out *Log In*.
2. Seller clicks the button to look over/update inventory.
3. Seller clicks the “add product” button.
4. Application opens a pop-up window with fields for the product’s characteristics.
5. Seller inputs characteristics of the product into their respective fields including name, invoice price, sell price, and available quantity.
6. Seller clicks the “confirm” button to finalize the addition.
7. Pop-up window is closed.
8. The new product is added to the list of items seen on the seller’s inventory window.

CRC Cards (Rafael)

Product	
• RESPONSIBILITIES • ---------- • Provides product details • Provides product quantity/availability	• COLLABORATORS • ---------- • Shopping Cart • Inventory

Shopping Cart	
• RESPONSIBILITIES • ---------- • Adds up total price of items • Manages the products selected in the user's cart	• COLLABORATORS • ---------- • Product • User

User	
• RESPONSIBILITIES • ---------- • Manages user profile to make changes • Manages user inventory to make changes	• COLLABORATORS • ---------- • Shopping Cart • Order

Order	
• RESPONSIBILITIES • ---------- • Manages user orders by using user's account information	• COLLABORATORS • ---------- • Product • User • Payment

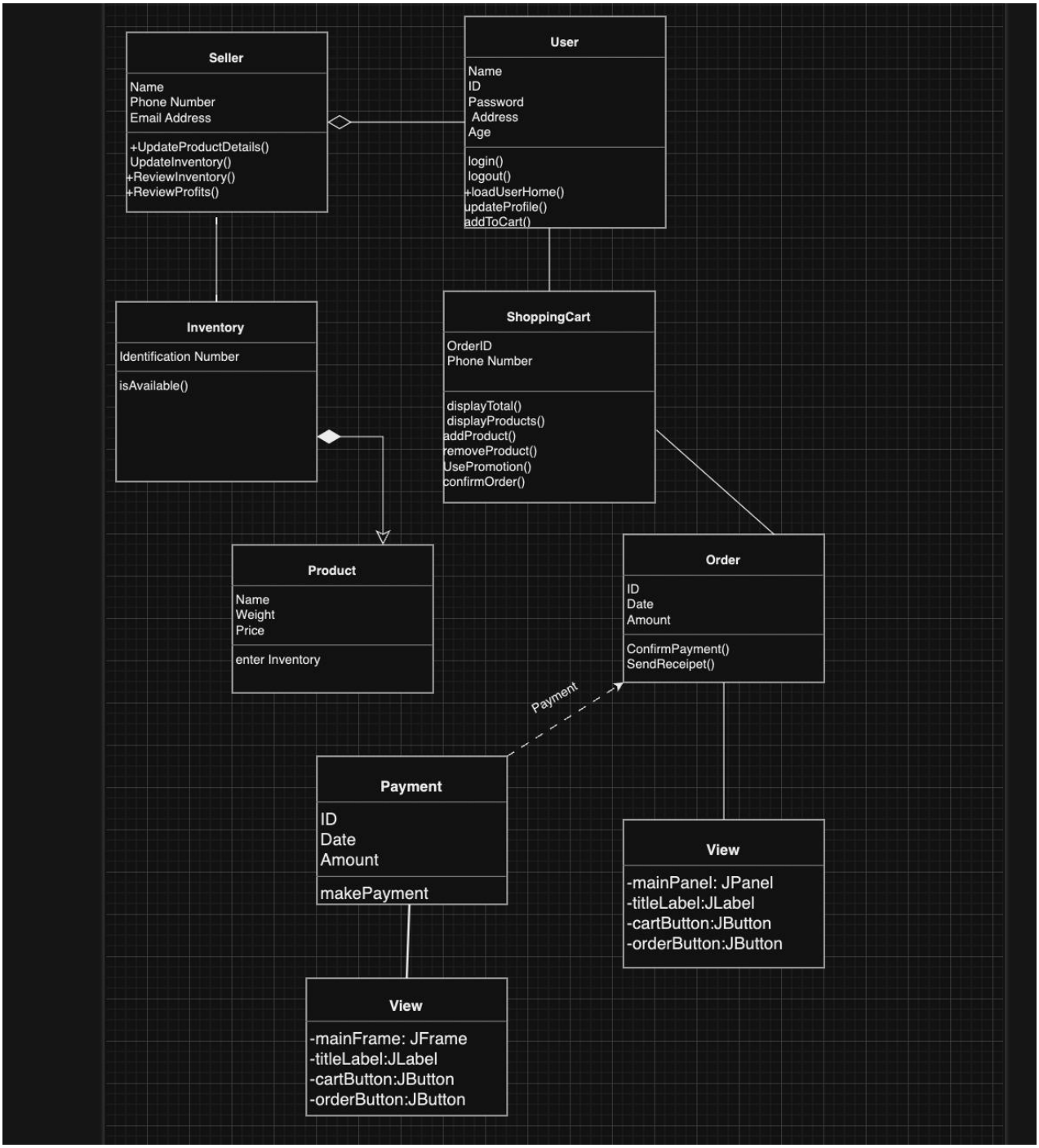
Inventory	
• RESPONSIBILITIES • ---------- • Manages available products to show customers if their is available stock	• COLLABORATORS • ---------- • Product • Seller

Seller	
• RESPONSIBILITIES • ---------- • Manages which products are available to sell	• COLLABORATORS • ---------- • Product • Inventory

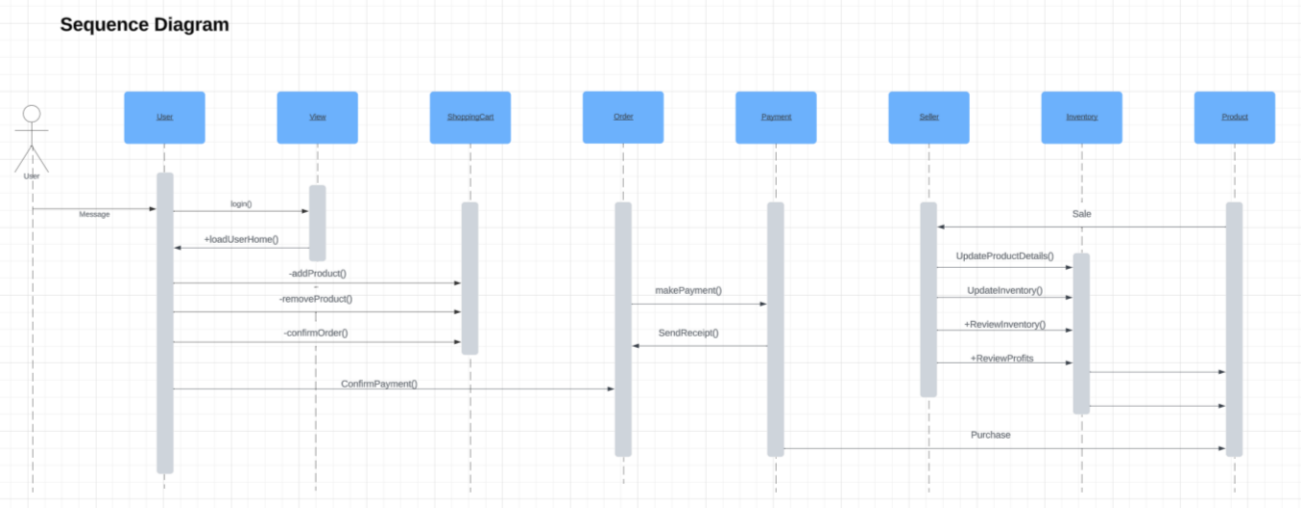
Payment	
• RESPONSIBILITIES • ---------- • Manages user's payment information to process • Checks to verify user's account and payment information	• COLLABORATORS • ---------- • User • Order

UML Diagrams

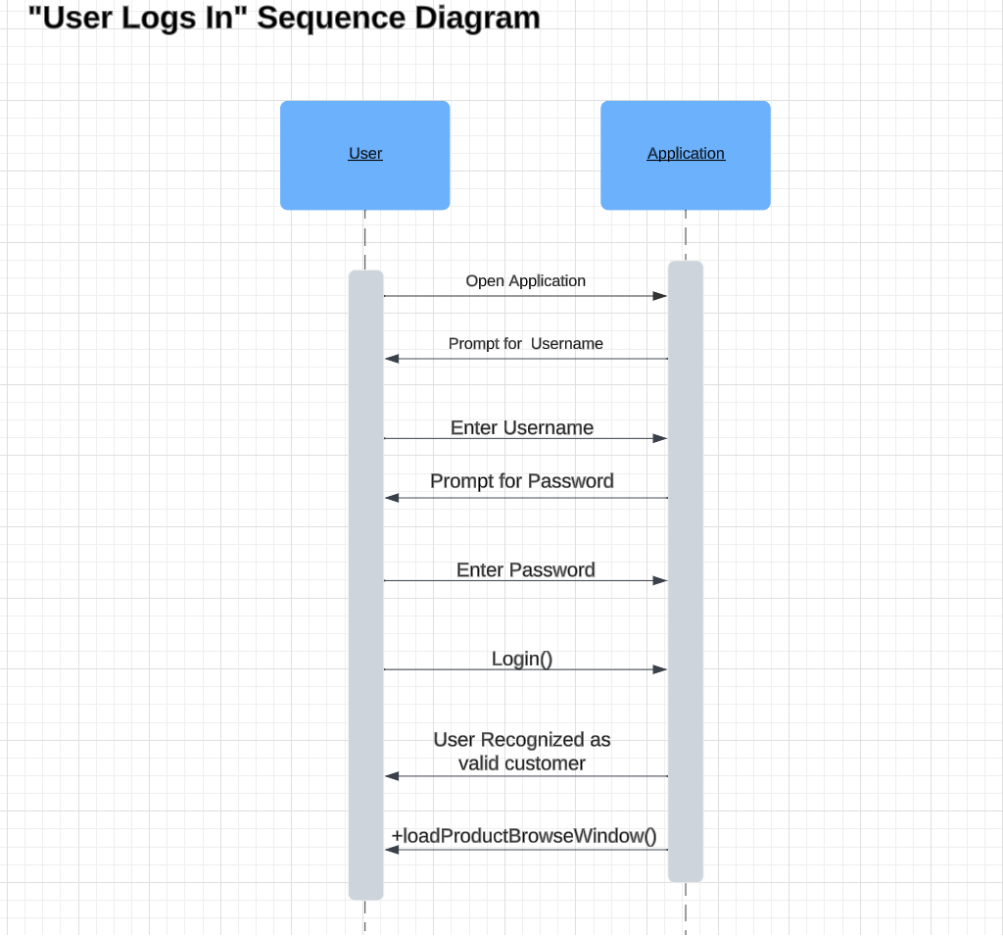
Class Diagram (John, Valente, Tivvon)



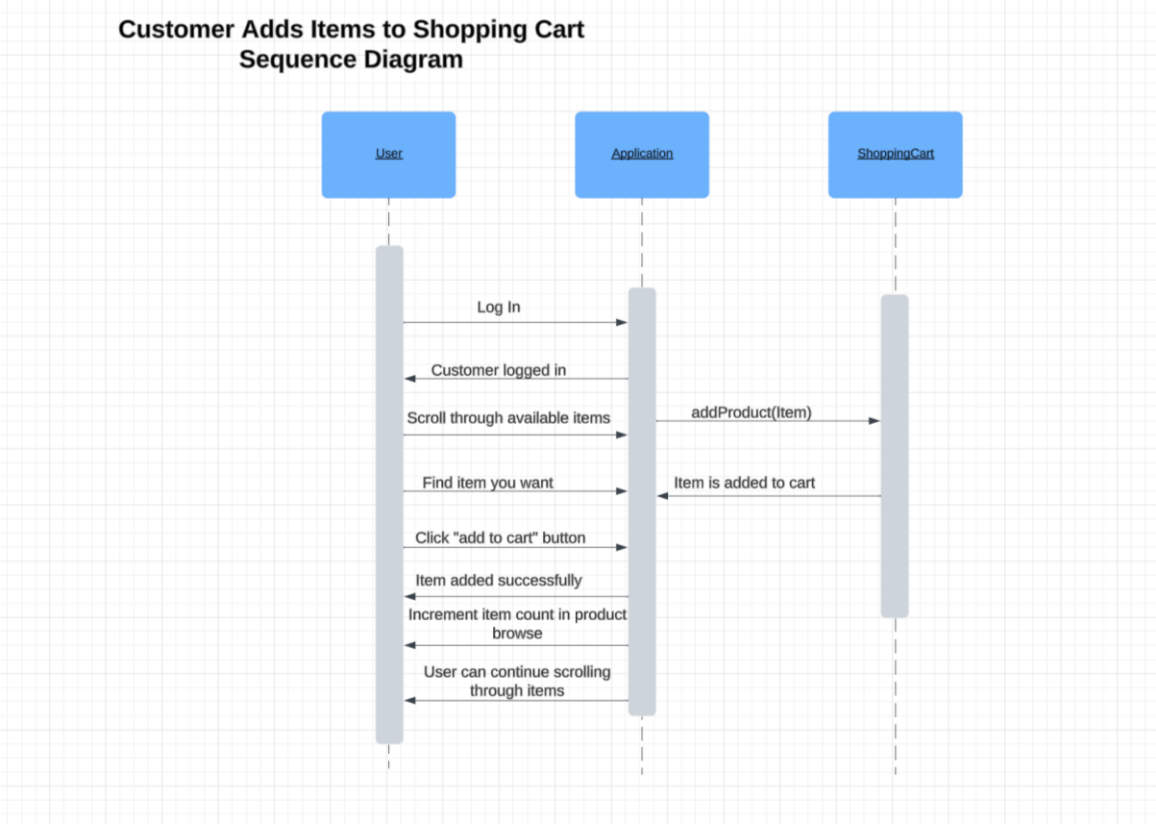
Sequence Diagrams (Rafael)



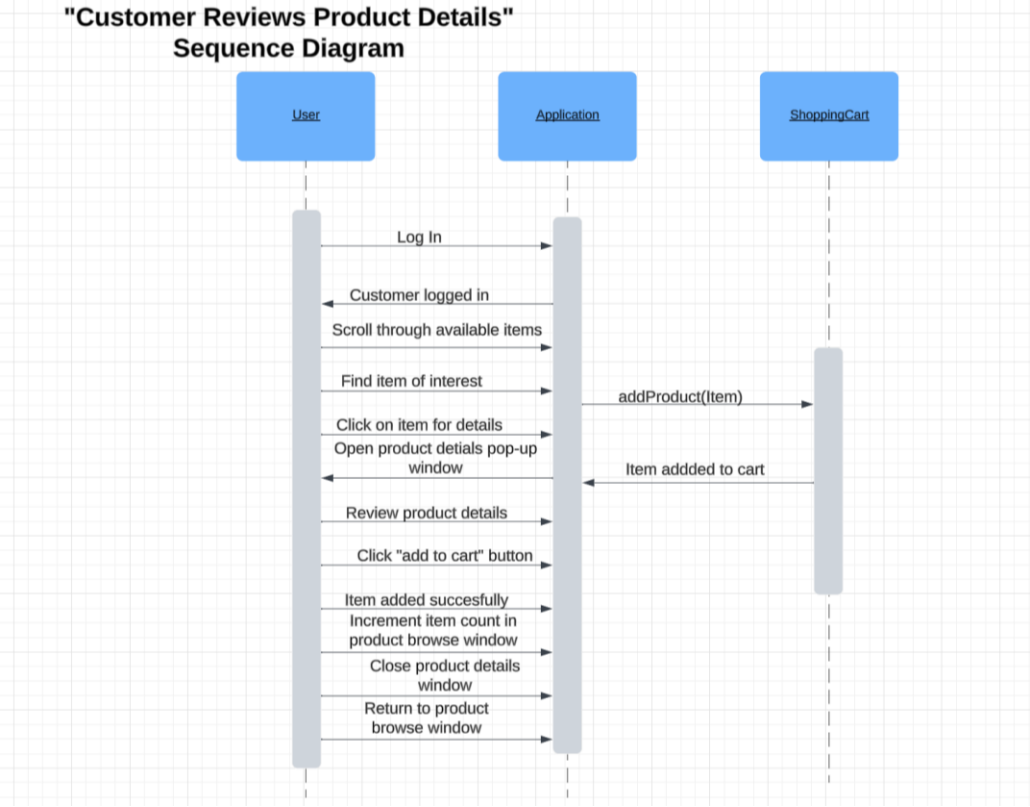
“User logs in” Sequence Diagram



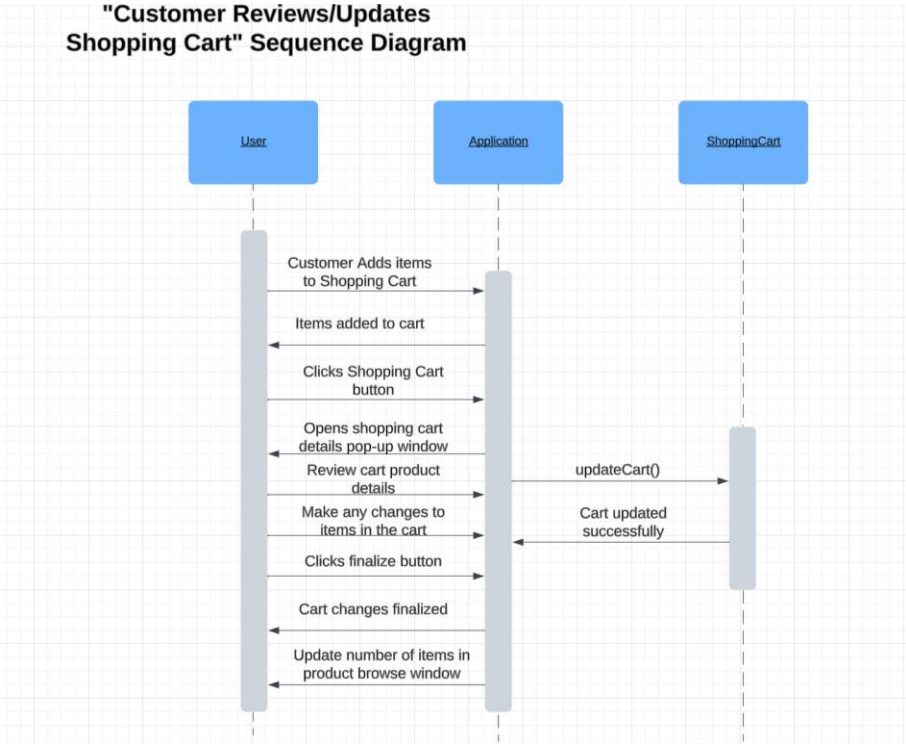
“Customer Adds Items To Shopping Cart” Sequence Diagram



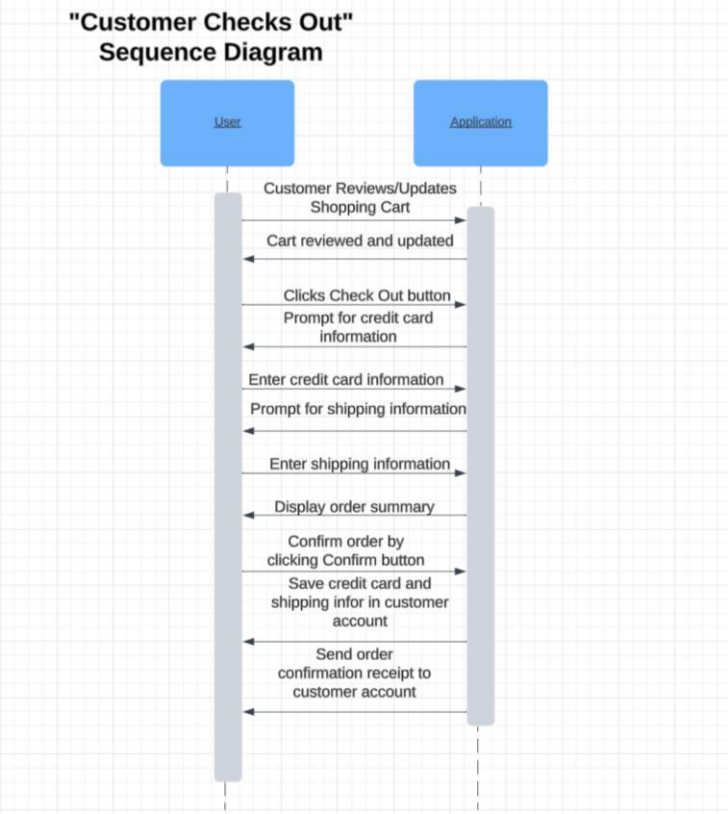
“Customer Reviews Product Details” Sequence Diagram



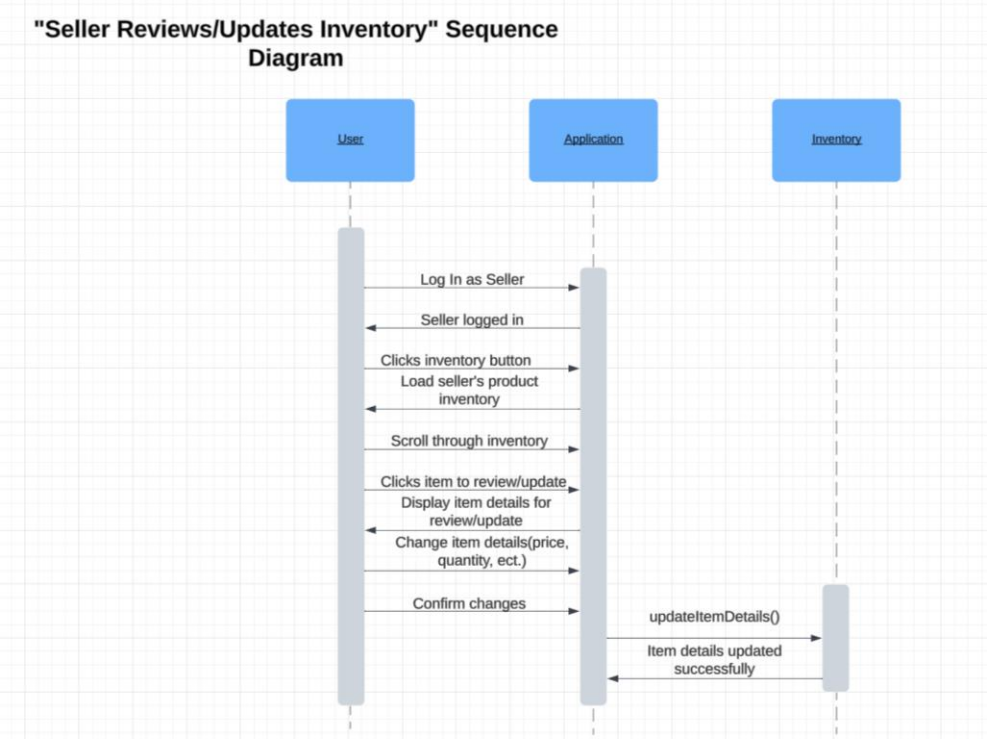
“Customer Reviews/Updates Shopping Cart” Sequence Diagram



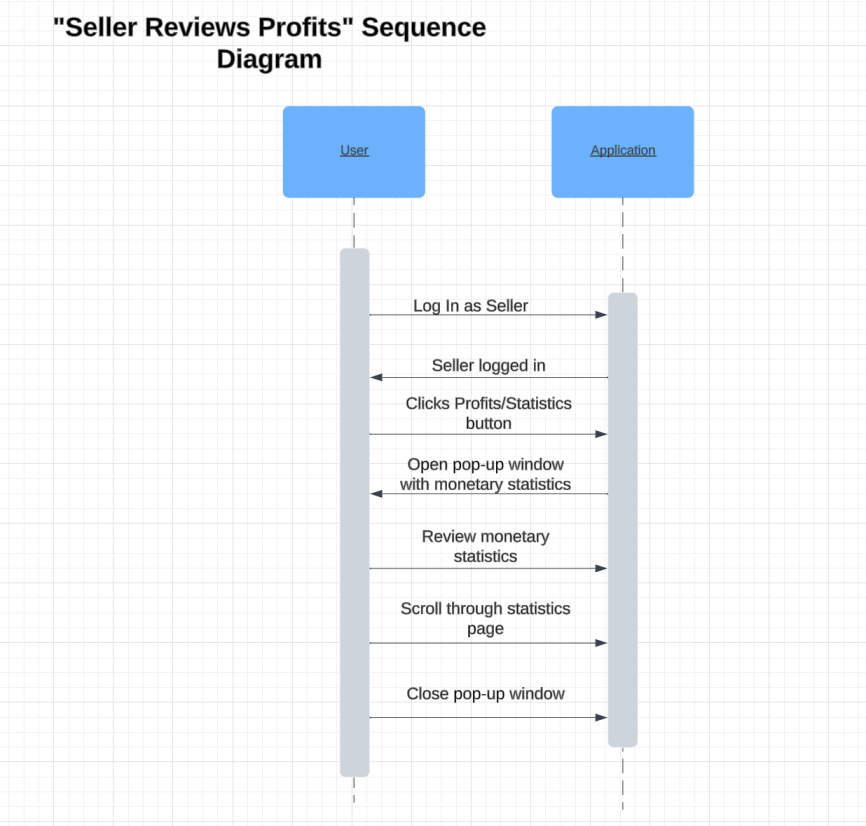
“Customer Checks Out” Sequence Diagram



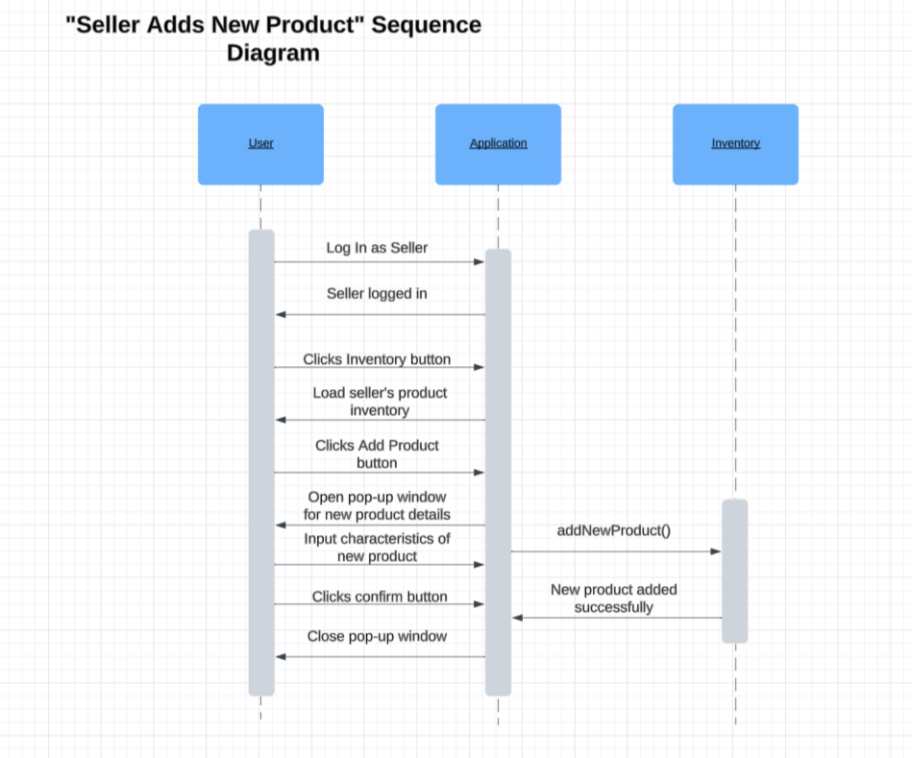
“Seller Reviews/Updates Inventory” Sequence Diagram



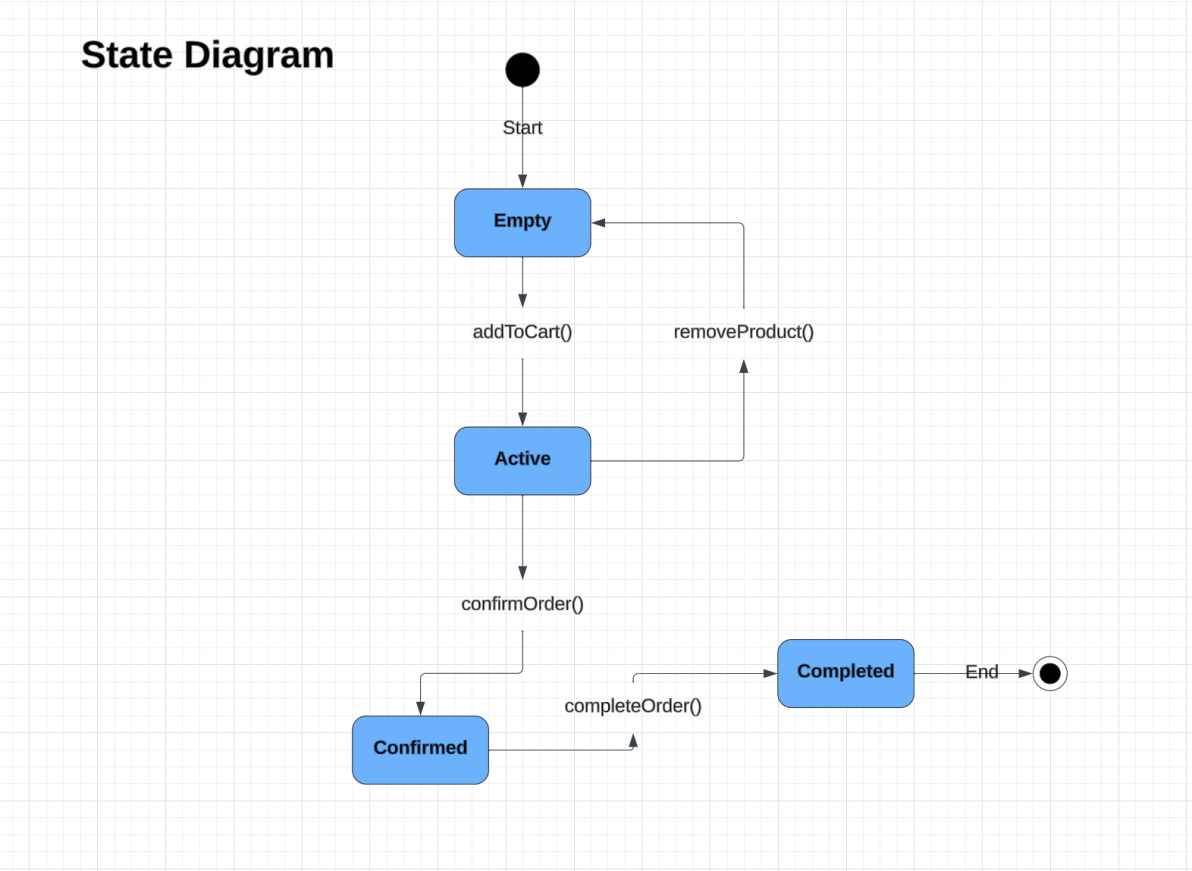
“Seller Reviews Profits” Sequence Diagram



“Seller Adds New Product” Sequence Diagram


"Seller Adds New Product" Sequence Diagram
User
Application
Inventory
Log In as Seller
Seller logged in
Clicks Inventory button
Load seller's product inventory
Clicks Add Product button
Open pop-up window for new product details
Input characteristics of new product
addNewProduct()
Clicks confirm button
New product added successfully
Close pop-up window

State Diagram (Rafael)



Source Code

Inventory Class (John)

```
Package com.mycompany.shoppingcart;

import java.util.ArrayList;
import java.util.Iterator;
public class Inventory {
    private ArrayList<Product> items;

    public Inventory(ArrayList<Product> i) {
        this.items = i;
    }

    public void add(Product p) {
        items.add(p);
    }

    public void remove(int id) {
        Iterator<Product> iterator = items.iterator();
        while (iterator.hasNext()) {
            Product p = iterator.next();
            if (p.getId() == id) {
                iterator.remove();
                break; // Exit the loop after removing the item
            }
        }
    }

    public Boolean isAvailable(int id) {
        for (Product p : items) {
            if (p.getId() == id) {
                return true;
            }
        }
        return false;
    }
}
```

Order Class (John)

```
package com.mycompany.shoppingcart;
import java.util.Date;

/**
 *
 *
 */
public class Order {
    public Order(int i) {
        this.id = i;
    }

    public void confirmPayment() {
        //placeholder
    }

    public void sendReceipt() {
        //placeholder
    }

    private int id;
    private Date date;
    private double amount;
}
```

Payment Class (John)

```
package com.mycompany.shoppingcart;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Date;

/**
 *
 * @]
 */
public class Payment {
    public Payment(int i, double a) {
        this.id = i;
        this.amount = a;

        Date d = new java.util.Date();
        this.date = d;
    }

    private int id;
    private Date date;
    private double amount;
}
```

Product Class (John, Valente)

```
package com.mycompany.shoppingcart;

/**
 * Product class represents a product with an ID, name, price, and quantity.
 */
public class Product {

    private int id;
    private String name;
    private double price;
    private int quantity;

    public Product(int id, String name, double price, int quantity) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public int getQuantity() {
        return quantity;
    }

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

Seller Class (John)

```
package com.mycompany.shoppingcart;

/**
 *
 *
 */
public class Seller extends User {
```

```
    public Seller (String n, String p, String a, String e, int i, int age, int phone) {
        super(n, p, a, e, i, age);
        this.phoneNumber = phone;
    }

    public void updateProductDetails() {
        //placeholder
    }

    public void updateInventory() {
        //placeholder
    }

    public void reviewInventory() {
        //placeholder
    }

    public void reviewProfits() {
        //placeholder
    }

    private int phoneNumber;
}
```

Shopping Cart Class (John, Valente)

```
package com.mycompany.shoppingcart;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import javax.swing.JOptionPane;

/**
 * ShoppingCart class represents the shopping cart functionality.
 */
public class ShoppingCart {

   private ArrayList<Product> items = new ArrayList<Product>();

/**
     * Display all products in the shopping cart.
     */
    public void showAllProducts(Product products[]) {
        System.out.println("Shopping Cart Contents:");
        System.out.printf("%-5s %-15s %-10s %-8s\n", "ID", "Name", "Price",
"Quantity");

        for (Product p : products) {
            System.out.printf("%-5d %-15s %-10.2f %-8d\n", p.getId(), p.getName(),
p.getPrice(), p.getQuantity());
        }
        System.out.println();
    }

    /**
     * Add a product to the shopping cart.
     *
     * @param product  Product to add.
     * @param quantity Quantity of the product to add.
     */
    public void addProduct(Product product, int quantity) {
        if (product != null && product.getQuantity() >= quantity) {
            product.setQuantity(product.getQuantity() - quantity);
```

```
            items.add(new Product(product.getId(), product.getName(),
product.getPrice(), quantity));
            System.out.println("Product added successfully");
        } else if (product != null) {
            System.out.println("Quantity is not present in the stock");
        } else {
            System.out.println("Product not found");
        }
        System.out.println();
    }

    /**
     * Remove a product from the shopping cart.
     *
     * @param id ID of the product to remove.
     * @param products Array of available products.
     */
    public void removeProduct(int id, Product products[]) {
        Product removedProduct = null;
        for (Product product : items) {
            if (product.getId() == id) {
                removedProduct = product;
                break;
            }
        }

        if (removedProduct != null) {
            items.remove(removedProduct);
            updateProductQuantity(products, id, removedProduct.getQuantity());
            System.out.println("Product removed from the cart");
        } else {
            System.out.println("Product not found in the cart");
        }

        System.out.println();
    }

    // Helper method to update product quantity in the products array
    private void updateProductQuantity(Product products[], int id, int quantity) {
```

```
        for (Product product : products) {
            if (product.getId() == id) {
                product.setQuantity(product.getQuantity() + quantity);
                break;
            }
        }
    }

    // ... (other methods remain unchanged)
}
```

ShoppingCartUI Class (Valente)

```
package com.mycompany.shoppingcart;

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

public class ShoppingCartUI extends JFrame {

    private ShoppingCart shoppingCart;

    public ShoppingCartUI(Product[] products) {
        shoppingCart = new ShoppingCart();

        setTitle("Shopping Cart");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton showProductsButton = new JButton("Show All Products");
        JButton addToCartButton = new JButton("Add to Cart");
        JButton deleteFromCartButton = new JButton("Delete from Cart");
        JButton editCartButton = new JButton("Edit Cart");
        JButton showTotalButton = new JButton("Show Total");

        showProductsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                shoppingCart.showAllProducts(products);
            }
        });

        addToCartButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int productId = Integer.parseInt(JOptionPane.showInputDialog("Enter
Product ID:"));
                int quantity = Integer.parseInt(JOptionPane.showInputDialog("Enter
Quantity:"));
                shoppingCart.addProduct(findProductById(productId, products), quantity);
```

```
        }
    });

    deleteFromCartButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            int productId = Integer.parseInt(JOptionPane.showInputDialog("Enter
Product ID to Delete:"));
            shoppingCart.removeProduct(productId, products);
        }
    });

    editCartButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Add logic for editing cart
        }
    });

    showTotalButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Add logic for showing total
        }
    });

    setLayout(new FlowLayout());

    add(showProductsButton);
    add(addToCartButton);
    add(deleteFromCartButton);
    add(editCartButton);
    add(showTotalButton);

    pack();
    setLocationRelativeTo(null);
    setVisible(true);
}
```

```
    // Helper method to find a product by ID
    private Product findProductById(int id, Product[] products) {
        for (Product product : products) {
            if (product.getId() == id) {
                return product;
            }
        }
        return null;
    }

    public static void main(String[] args) {
        Product[] products = new Product[5];
        products[0] = new Product(101, "Juice", 102.50, 5);
        products[1] = new Product(102, "Chocolates", 402.50, 14);
        products[2] = new Product(103, "Shop", 42.80, 8);
        products[3] = new Product(104, "Biscuits", 10.00, 55);
        products[4] = new Product(105, "Candy", 12.20, 102);

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new ShoppingCartUI(products);
            }
        });
    }
}
```

User Class (John)

```
package com.mycompany.shoppingcart;

/**
 *
 */
public class User {
    private String name;
    private String password;
    private String address;
    private String email;
    private int id;
    private int age;
    private ShoppingCart cart;

    public User(String n, String p, String a, String e, int i, int age) {
        this.name = n;
        this.password = p;
        this.address = a;
        this.email = e;
        this.id = i;
        this.age = age;
        this.cart = new ShoppingCart();
    }


    public ShoppingCart getCart() {
        return cart;
    }
}
```

Main Class (Valente)

```
package com.mycompany.shoppingcart;
import java.util.HashMap;
import java.util.Map;
import javax.swing.SwingUtilities;

import java.util.*;
/**
 *
 * @author Valente Lowery
 */

public class Main

/ To show the products available in the store
    public static void main(String args[]) {
    // Scanner class is used to take the input from the user (or console)
    Scanner scan = new Scanner(System.in);

    // To store the product details
    Product products[] = new Product[5];
    // Adding the product details
    products[0] = new Product(101, "Juice", 102.50, 5);
    products[1] = new Product(102, "Chocolates", 402.50, 14);
    products[2] = new Product(103, "Shop", 42.80, 8);
    products[3] = new Product(104, "Biscuits", 10.00, 55);
    products[4] = new Product(105, "Candy", 12.20, 102);

     SwingUtilities.invokeLater(() -> new ShoppingCartUI(products));

    // Authentication of the customer
    System.out.println("Enter your username to login");
    String username = scan.nextLine();
    System.out.println("Enter your password to login");
    String password = scan.nextLine();

    // While the customer does not enter the correct id and password, the system will
not be accessible
    while (!username.equals("User") || !password.equals("1234")) {
```

```
        System.out.println("Invalid credentials. Enter your username and password to
login");
        System.out.print("Enter your username: ");
        username = scan.nextLine();
        System.out.print("Enter your password: ");
        password = scan.nextLine();
    }

    // To store the items that the customer wants to buy
     HashMap<Integer, Integer> cart = new HashMap<>();
        ShoppingCart shoppingCart = new ShoppingCart(); // Create an instance of
ShoppingCart
        int choice;

        do {
            System.out.println("1. Show all products");
            System.out.println("2. Add product to cart");
            System.out.println("3. Delete product from cart");
            System.out.println("4. Edit product quantity in cart");
            System.out.println("5. Show total price");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = scan.nextInt();
            scan.nextLine(); // To read the data from the next line from the console

            switch (choice) {
                case 1:
                    shoppingCart.showAllProducts(products);
                    break;
                case 2:
                    System.out.print("Enter the product id : ");
                    int addProductId = scan.nextInt();
                    scan.nextLine();
                    System.out.print("Enter the quantity needed : ");
                    int addProductQuantity = scan.nextInt();
                    scan.nextLine();
                    shoppingCart.addProduct(products, cart, addProductId,
addProductQuantity);
                    break;
```

```
            case 3:
                System.out.print("Enter the product id to delete: ");
                int deleteProductId = scan.nextInt();
                scan.nextLine();
                shoppingCart.removeProduct(cart, deleteProductId);;
                break;
            case 4:
                System.out.print("Enter the product id : ");
                int editProductId = scan.nextInt();
                scan.nextLine();
                System.out.print("Enter the new quantity needed : ");
                int editProductQuantity = scan.nextInt();
                scan.nextLine();
                shoppingCart.editProduct(products, cart, editProductId,
editProductQuantity);
                break;
            case 5:
                shoppingCart.totalPrice(products, cart);
                break;
            case 6:
                shoppingCart.confirmOrder();
                System.out.println("Exiting. Thank you!");
                break;
            default:
                System.out.println("Invalid choice. Please enter a valid option.");
        }
    } while (choice != 6);
  }
}
```