COP4331
11/30/24

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

Shopping Cart Application

Functional Specification

The Shopping Cart application will be implemented using Java Swing for the graphical user interface and Jackson for storing the product and user databases persistently with JSON text files. When the application is closed and reopened, the state of the product inventory and user data will be restored. The initial states of the JSON data files for users and products will be pre-populated with sample data for demo/testing purposes.

The application allows two types of users—Customers and Sellers—to log in and perform different tasks related to shopping and inventory management. The application maintains products, allows for customer purchases, and provides sellers with financial information such as costs, revenues, and profits.

Customers will be able to:

- View a list of available products along with their prices and available quantities.
- Add products or product bundles to their shopping cart with a specified quantity.
- View their cart and modify quantities or remove items before checking out.
- Check out the items in their cart by inputting their credit card info.

Sellers will be able to:

- View their product inventory, including product name, ID, type, quantity, invoice price, and selling price.
- Add new products, specifying details like invoice price, selling price, and quantity.
- Add discounts to products.
- Create product bundles.
- View financial data including revenues, costs, and profits.

Upon launching the application, a login window will appear. The user inputs a username and password, and the system will validate the credentials with the database. Depending on if the user type is a seller or customer, the appropriate window will be displayed with the different functionalities available.

The customer window will show a list of available products to purchase. The customer can add products or product bundles to their cart by inputting the ID and quantity desired. A customer will not be able to input a quantity greater than the available stock. The current cart contents will be displayed in the main customer view, but the customer can click to view their cart to see the total price, update quantities, or remove items. When checking out, the customer can submit their credit card information or cancel; the payment process is abstracted, and all payments are assumed to be successful. After a checkout is completed, the purchased products' stock is updated accordingly, as well as the seller's financial data.

COP4331
11/30/24

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

The seller window will show a list of the seller's inventory. The seller can add products to their inventory by inputting the product name, description, invoice price, and selling price. There will also be buttons for the seller to create product bundles, apply a discount to a product, and view their financial data. To create a product bundle, a pop up window will prompt the seller to select at least two products. A 10% discount will be applied to the bundle and it will be added to the seller's inventory. To apply a discount to a product, a pop up window will prompt the seller to select a product and input the discount rate (i.e. .2 for 20% off), then the discount will be applied to that product. Any product in the sellers inventory would be visible in the list of available products for customers to purchase in the customer view.

Use Cases

User Logs In

1. System displays log in window.
2. User enters username and password.
3. System validates the credentials belonging to a Customer.
4. System displays the customer view (product browsing) window.

Variation #1: User is a Seller

1.1. Start at Step 2.
1.2. System validates the credentials belonging to a Seller.
1.3. System displays the seller view (inventory management) window.

Variation #2: Incorrect credentials

1.4. Start at Step 2.
1.5. System displays an error message saying the login credentials are invalid.
1.6. System prompts user to retry the login.

Customer Adds Item to Cart

1. Customer inputs the desired product ID and quantity, and clicks "Add to Cart" button.
2. System adds item to cart, updating total price and item count.
3. System displays message saying X item(s) was added to cart.

Variation #1: Not enough item stock

1.1. Start at Step 1.
1.2. System displays error message saying there is not enough stock for item to be added to cart.

COP4331
11/30/24

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

Customer Reviews/Updates Cart

1. Customer clicks cart icon to view it.
2. System opens pop up to display contents of the cart, including the product list, quantities, and total price.
3. Customer updates the quantity of a product.
4. System updates the cart with the new quantity.
5. System displays the updated cart, including new total price.

Customer Checks Out

1. Customer clicks “Checkout” button.
2. System opens checkout pop up window.
3. Customer inputs credit card information and clicks “Submit”.
4. System processes payment successfully.
5. System decreases the product(s)’ stock and updates seller financial data to reflect the purchase.
6. System displays the confirmation to the customer.

Variation #1: Customer cancels checkout

1.1. Start at Step 2.
1.2. Customer clicks “Cancel” button.
1.3. System closes checkout window.

Seller Reviews/Updates Inventory

1. System displays within inventory management window the current inventory list including each product name, type, quantity, invoice price, and selling price.
2. Seller scrolls through the inventory list to see all products and their information.

Seller Adds New Product

1. Seller clicks option to add a new product within the seller view window.
2. System displays a form to enter the new product details (name, invoice price, sell price, quantity).
3. Seller enters the product name, invoice price, sell price, and available quantity.
4. System adds the new product to the inventory and updates financial data (cost).
5. System displays confirmation that the product was added.
6. System updates product list for Customers to view and purchase.

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

Seller Adds Product Discount

1. Seller clicks button to apply a product discount within the seller view.
2. System displays a pop up prompting Seller to select an item.
3. Seller selects product to apply discount to.
4. System prompts seller to input discount rate as a decimal.
5. Seller inputs discount rate and submits.
6. System applies the discount rate to the product selling price.

Seller Creates Product Bundle

1. Seller clicks button to create a product bundle.
2. System displays a pop up prompting Seller to select at least 2 items.
3. Seller selects items for the bundle.
4. System adds the product bundle to the inventory list and with a 10% discounted total price of the items.

Seller Views Financial Data

1. Seller clicks “View Financial Data” button.
2. System displays financial data window which shows a summary of revenues, costs, and profits.

Glossary

Customer: A user who accesses the application to browse products, add items to their shopping cart, and complete purchases.

Seller: A user who manages the inventory, adds new products, updates existing product details, and tracks financial data related to sales, costs, and profits.

Cart: A virtual container where customers can temporarily store products they wish to purchase before proceeding to checkout.

Inventory: The collection of products available for sale in the application, including details such as product ID, name, type, quantity, invoice price, and selling price.

Invoice Price: The price at which a seller acquires a product for resale.

Selling Price: The price at which a product is offered to customers for purchase.

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

Product: An item available for sale within the application, which includes details such as name, type, price, and quantity available.

Checkout: The process a customer goes through to complete a purchase, including reviewing the shopping cart and providing payment information.

Persistence: The ability of the application to save user and product data in such a way that it is retained even after the application is closed and reopened. This is achieved through text files or Java Serialization.

Java Serialization: A process in Java that converts an object into a byte stream for storage or transmission, allowing for the persistence of data across sessions.

Swing: A Java toolkit used for building graphical user interfaces (GUIs) in the application, enabling interaction between users and the system.

Jackson: A Java library used to serialize/map java objects to JSON and vice versa.

Revenue: The total amount of money earned by the seller from selling products.

Costs: The amount of money spent by the seller to acquire products for sale, typically the sum of the invoice prices.

Profit: The financial gain realized by the seller, calculated as the difference between revenue and costs.

COP4331
11/30/24

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

Design Specification

CRC Cards:

System

Responsibilities:

Login Functionality:
-Display the login window.
-Validate the credentials of users (Customer or Seller).
-Direct valid Customer to the product browsing window.
-Direct valid Seller to the inventory management window.
-Handle incorrect login attempts and display error messages.

Reviewing Product Details:
-Retrieve the selected product's information (description, price, availability).
-Display the product details in a pop-up window for the Customer to review.

Adding Item to Cart:
-Add the selected product to the customer's cart with the specified quantity.
-Update the cart total and item count.
-Display a confirmation message that the cart has been updated.
-Handle the error if the product is unavailable.

Reviewing/Updating Cart:
-Retrieve and display the contents of the cart, including product list, quantities, and total price.
-Allow the Customer to update product quantities.
-Update the cart with the new quantities and recalculate the total price.
-Handle the case where the cart is empty and display a message.
-Display the updated cart and total price to the Customer.

Checking Out:
-Prompt the Customer to enter credit card information.
-Process the payment and confirm the purchase.
-Update product inventory to reflect purchased items.
-Display the order confirmation and details.

Reviews/Updates Inventory:
-Display the current inventory, including product name, ID, type, quantity, invoice price, and selling price.
-Retrieve and display the details of the selected product for the Seller to update.
-Save the updated inventory information after the Seller makes changes.
-Display a confirmation message that the inventory was updated.

Adds New Product:
-Display a form for the Seller to enter new product details (name, invoice price, sell price, and quantity).
-Add the new product to the inventory list once the Seller submits the details.
-Display confirmation that the new product has been successfully added to the inventory.
-Update the product list for Customers to view and purchase.

Views Financial Data:
-Retrieve financial data, including revenues, costs, and profits from completed sales and current inventory.
-Display the financial data by each product in the inventory.

Collaborators:
Customer
Seller
Database
Product
Cart
FinancialData

Database

Responsibilities:

Login Functionality:
-Store user credentials (both Customer and Seller).
-Retrieve and verify credentials for login.
-Return user type (Customer or Seller) based on credentials provided.

Reviewing Product Details:
-Store and retrieve product details such as name, description, price, and availability.

Adding Item to Cart:
-Retrieve product information such as name, price, and availability.
-Store customer cart data, including selected products and quantities.
-Update inventory for products after they are added to the cart or purchased (this will likely be handled during checkout but should be considered for future functionality).
-Maintain product data for Customer browsing and Seller inventory management.

Reviewing/Updating Cart:
-Retrieve and store cart information (list of products, quantities, and total price).
-Update the cart data when the Customer modifies product quantities.

Checking Out:
-Process payment information.
-Update inventory data to reflect purchased items.
-Clear cart data after successful checkout.

Reviews/Updates Inventory:
-Store and retrieve product details such as name, ID, type, quantity, invoice price, and selling price.
-Save the updated inventory details after a Seller makes changes to a product.

Adds New Product:
-Store new product details (name, invoice price, sell price, quantity) after a Seller adds the product.
-Update inventory so the new product is available for Customers to view and purchase.

Views Financial Data:
-Retrieve and store financial data such as revenues, costs, and profits from completed sales.
-Retrieve current inventory data and associate it with financial reports for each product.

Collaborators:
System
Customer
Seller
Product
Cart
FinancialData

Customer

Responsibilities:

Login Functionality:
-Provide username and password for login.
-Access the product browsing window if credentials are valid.
-Retry login if credentials are incorrect.

Reviewing Product Details:
-Click on a product to view its details (description, price, availability).

Adding Item to Cart:
-Select a product to add to the cart.
-Specify the quantity of the product to add.
-View confirmation of the updated cart.

Reviewing/Updating Cart:
-Navigate to the checkout window to review the cart.
-Update the quantity of products in the cart.
-Review the updated cart and total price.
-Handle case where cart is empty.

Checking Out:
-Click to proceed to checkout.
-Enter credit card information.
-Complete the purchase and view order confirmation.

Collaborators:
System
Database
Product
Cart

Cart

Responsibilities:

Adding Item to Cart:
-Select a product to add to the cart.
-Specify the quantity of the product to add.
-View confirmation of the updated cart.

Reviewing/Updating Cart:
-Retrieve the list of products in the cart, including quantities and total price.
-Update the quantity of a product based on the Customer's input.
-Recalculate the total price based on updated quantities.
-Handle the case where the cart is empty and return an empty cart state.

Checking Out:
-Convert cart contents into an Order for processing.
-Clear the cart once the purchase is completed.

Collaborators:
Customer
System
Product
Database
FinancialData

Seller

Responsibilities:

Login Functionality:
-Provide username and password for login.
-Access the inventory management window if credentials are valid.
-Retry login if credentials are incorrect.

Reviews/Updates Inventory:
-View the current inventory list, including product details such as name, ID, type, quantity, invoice price, and selling price.
-Select a product to update.
-Update the product's quantity and/or price.
-Review confirmation that the inventory was updated.

Adds New Product:
-Click the option to add a new product in the inventory management window.
-Enter product details like name, invoice price, sell price, and available quantity.
-Confirm the new product has been added to the inventory.

Views Financial Data:
-Navigate to the financial data section.
-View financial data, including revenues, costs, and profits from completed sales and current inventory.

Collaborators:
System
Product
Database
FinancialData

Product

Responsibilities:

Adding Item to Cart:
-Provide product information such as name, price, and availability.
-Check availability to ensure the product can be added to the cart.

Reviewing Product Details:
-Provide product information such as description, price, and availability when requested by the System.

Reviewing/Updating Cart:
Provide updated product details (price and availability) when the cart is modified or reviewed.

Checking Out:
-Update inventory to reflect the purchased quantities.

Reviews/Updates Inventory:
-Provide product information such as name, ID, type, quantity, invoice price, and selling price.
-Update the product's quantity and price based on Seller input.

Adds New Product:
-Store new product details such as name, invoice price, sell price, and available quantity.
-Add the new product to the inventory so it is available for Customers.

Views Financial Data:
-Track and provide financial data (revenues, costs, and profits) for each product.
-Provide inventory data related to financials, including product sales and current stock.

Collaborators:
System
Customer
Seller
Cart
Database
FinancialData

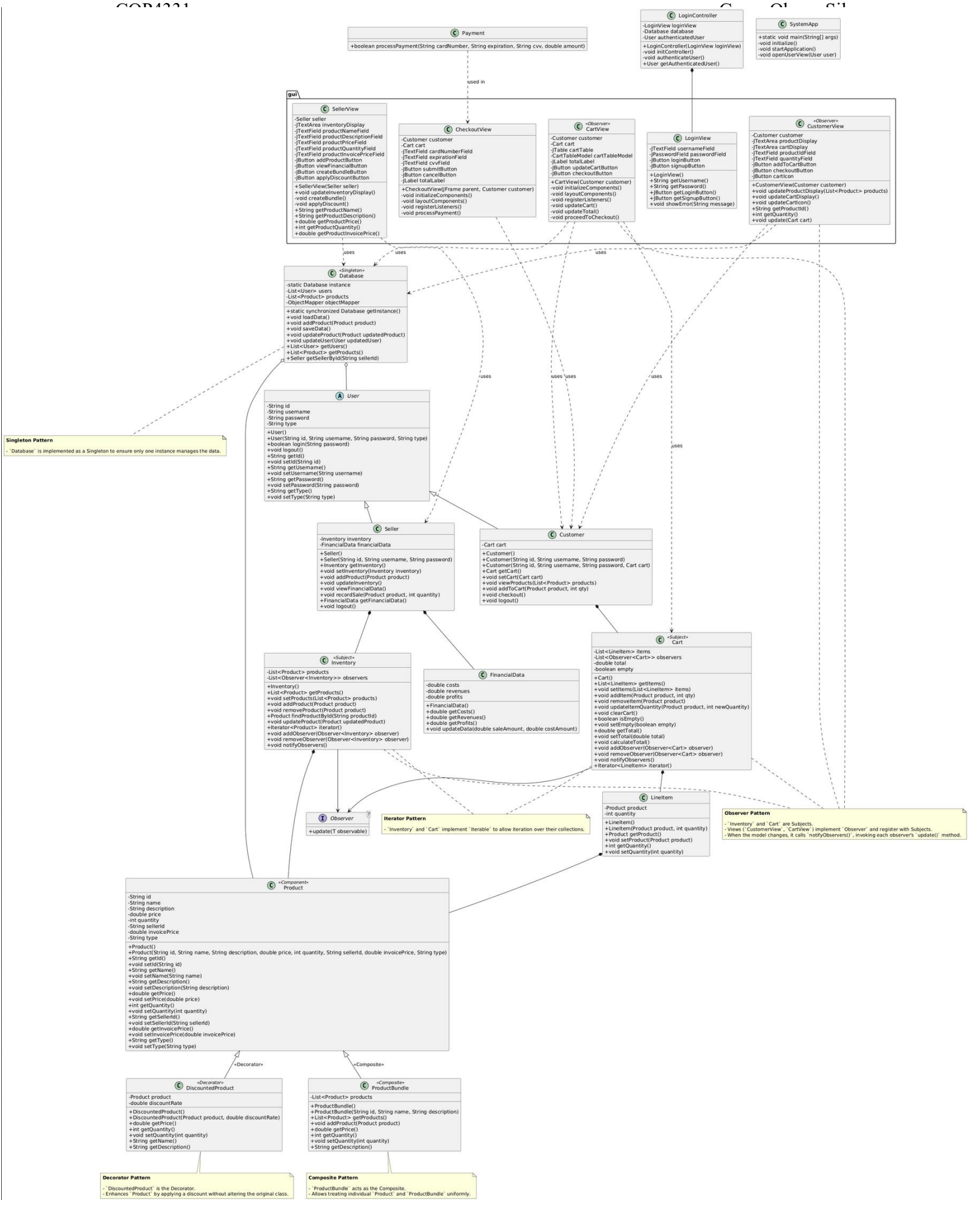
FinancialData

Responsibilities:

Adding Item to Cart:
-Calculate revenues, costs, and profits for each product.
-Store and update financial data related to completed sales.
-Provide summary data for Seller's financial reports.

Collaborators:
System
Product
Database

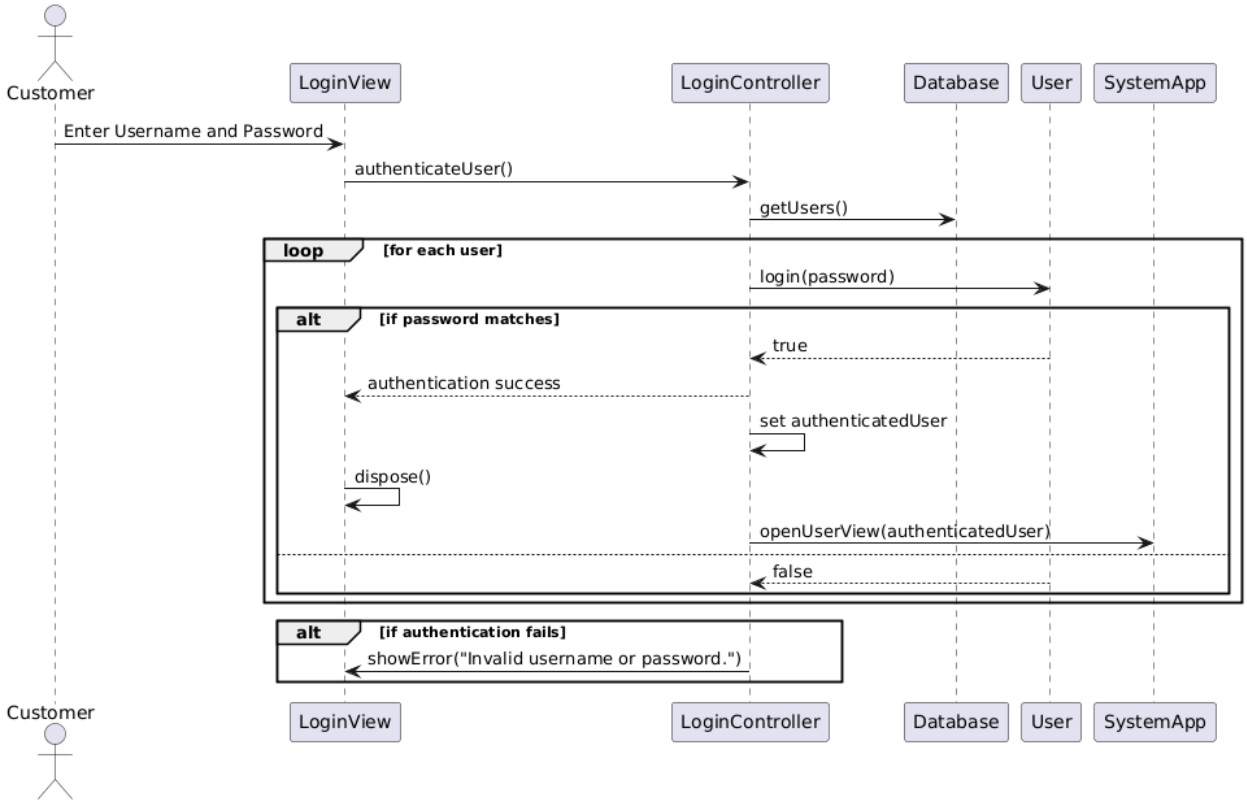
UML Class Diagram:


Payment
+boolean processPayment(String cardNumber, String expiration, String cvv, double amount)
used in
LoginController
-LoginView loginView
-Database database
-User authenticatedUser
+LoginController(LoginView loginView)
-void initController()
-void authenticateUser()
+User getAuthenticatedUser()
SystemApp
+static void main(String[] args)
-void initialize()
-void startApplication()
-void openUserView(User user)
gui
SellerView
-Seller seller
-JTextArea inventoryDisplay
-JTextField productNameField
-JTextField productDescriptionField
-JTextField productPriceField
-JTextField productQuantityField
-JTextField productInvoicePriceField
-JButton addProductButton
-JButton viewFinancialButton
-JButton createBundleButton
-JButton applyDiscountButton
+SellerView(Seller seller)
+void updateInventoryDisplay()
-void createBundle()
-void applyDiscount()
+String getProductName()
+String getProductDescription()
+double getProductPrice()
+int getProductQuantity()
+double getProductInvoicePrice()
CheckoutView
-Customer customer
-Cart cart
-JTextField cardNumberField
-JTextField expirationField
-JTextField cvvField
-JButton submitButton
-JButton cancelButton
-JLabel totalLabel
+CheckoutView(JFrame parent, Customer customer)
-void initializeComponents()
-void layoutComponents()
-void registerListeners()
-void processPayment()
«Observer»
CartView
-Customer customer
-Cart cart
-JTable cartTable
-CartTableModel cartTableModel
-JLabel totalLabel
-JButton updateCartButton
-JButton checkoutButton
+CartView(Customer customer)
-void initializeComponents()
-void layoutComponents()
-void registerListeners()
-void updateCart()
-void updateTotal()
-void proceedToCheckout()
LoginView
-JTextField usernameField
-JPasswordField passwordField
-JButton loginButton
-JButton signupButton
+LoginView()
+String getUsername()
+String getPassword()
+JButton getLoginButton()
+JButton getSignupButton()
+void showError(String message)
«Observer»
CustomerView
-Customer customer
-JTextArea productDisplay
-JTextArea cartDisplay
-JTextField productIdField
-JTextField quantityField
-JButton addToCartButton
-JButton checkoutButton
-JButton cartIcon
+CustomerView(Customer customer)
+void updateProductDisplay(List<Product> products)
+void updateCartDisplay()
+void updateCartIcon()
+String getProductId()
+int getQuantity()
+void update(Cart cart)
uses
«Singleton»
Database
-static Database instance
-List<User> users
-List<Product> products
-ObjectMapper objectMapper
+static synchronized Database getInstance()
+void loadData()
+void addProduct(Product product)
+void saveData()
+void updateProduct(Product updatedProduct)
+void updateUser(User updatedUser)
+List<User> getUsers()
+List<Product> getProducts()
+Seller getSellerById(String sellerId)
Singleton Pattern
- `Database` is implemented as a Singleton to ensure only one instance manages the data.
User
-String id
-String username
-String password
-String type
+User()
+User(String id, String username, String password, String type)
+boolean login(String password)
+void logout()
+String getId()
+void setId(String id)
+String getUsername()
+void setUsername(String username)
+String getPassword()
+void setPassword(String password)
+String getType()
+void setType(String type)
Seller
-Inventory inventory
-FinancialData financialData
+Seller()
+Seller(String id, String username, String password)
+Inventory getInventory()
+void setInventory(Inventory inventory)
+void addProduct(Product product)
+void updateInventory()
+void viewFinancialData()
+void recordSale(Product product, int quantity)
+FinancialData getFinancialData()
+void logout()
Customer
-Cart cart
+Customer()
+Customer(String id, String username, String password)
+Customer(String id, String username, String password, Cart cart)
+Cart getCart()
+void setCart(Cart cart)
+void viewProducts(List<Product> products)
+void addToCart(Product product, int qty)
+void checkout()
+void logout()
«Subject»
Inventory
-List<Product> products
-List<Observer<Inventory>> observers
+Inventory()
+List<Product> getProducts()
+void setProducts(List<Product> products)
+void addProduct(Product product)
+void removeProduct(Product product)
+Product findProductById(String productId)
+void updateProduct(Product updatedProduct)
+Iterator<Product> iterator()
+void addObserver(Observer<Inventory> observer)
+void removeObserver(Observer<Inventory> observer)
+void notifyObservers()
FinancialData
-double costs
-double revenues
-double profits
+FinancialData()
+double getCosts()
+double getRevenues()
+double getProfits()
+void updateData(double saleAmount, double costAmount)
«Subject»
Cart
-List<LineItem> items
-List<Observer<Cart>> observers
-double total
-boolean empty
+Cart()
+List<LineItem> getItems()
+void setItems(List<LineItem> items)
+void addItem(Product product, int qty)
+void removeItem(Product product)
+void updateItemQuantity(Product product, int newQuantity)
+void clearCart()
+boolean isEmpty()
+void setEmpty(boolean empty)
+double getTotal()
+void setTotal(double total)
+void calculateTotal()
+void addObserver(Observer<Cart> observer)
+void removeObserver(Observer<Cart> observer)
+void notifyObservers()
+Iterator<LineItem> iterator()
Observer
+update(T observable)
Iterator Pattern
- `Inventory` and `Cart` implement `Iterable` to allow iteration over their collections.
LineItem
-Product product
-int quantity
+LineItem()
+LineItem(Product product, int quantity)
+Product getProduct()
+void setProduct(Product product)
+int getQuantity()
+void setQuantity(int quantity)
Observer Pattern
- `Inventory` and `Cart` are Subjects.
- Views (`CustomerView`, `CartView`) implement `Observer` and register with Subjects.
- When the model changes, it calls `notifyObservers()`, invoking each observer's `update()` method.
«Component»
Product
-String id
-String name
-String description
-double price
-int quantity
-String sellerId
-double invoicePrice
-String type
+Product()
+Product(String id, String name, String description, double price, int quantity, String sellerId, double invoicePrice, String type)
+String getId()
+void setId(String id)
+String getName()
+void setName(String name)
+String getDescription()
+void setDescription(String description)
+double getPrice()
+void setPrice(double price)
+int getQuantity()
+void setQuantity(int quantity)
+String getSellerId()
+void setSellerId(String sellerId)
+double getInvoicePrice()
+void setInvoicePrice(double invoicePrice)
+String getType()
+void setType(String type)
«Decorator»
«Composite»
«Decorator»
DiscountedProduct
-Product product
-double discountRate
+DiscountedProduct()
+DiscountedProduct(Product product, double discountRate)
+double getPrice()
+int getQuantity()
+void setQuantity(int quantity)
+String getName()
+String getDescription()
«Composite»
ProductBundle
-List<Product> products
+ProductBundle()
+ProductBundle(String id, String name, String description)
+List<Product> getProducts()
+void addProduct(Product product)
+double getPrice()
+int getQuantity()
+void setQuantity(int quantity)
+String getDescription()
Decorator Pattern
- `DiscountedProduct` is the Decorator.
- Enhances `Product` by applying a discount without altering the original class.
Composite Pattern
- `ProductBundle` acts as the Composite.
- Allows treating individual `Product` and `ProductBundle` uniformly.

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

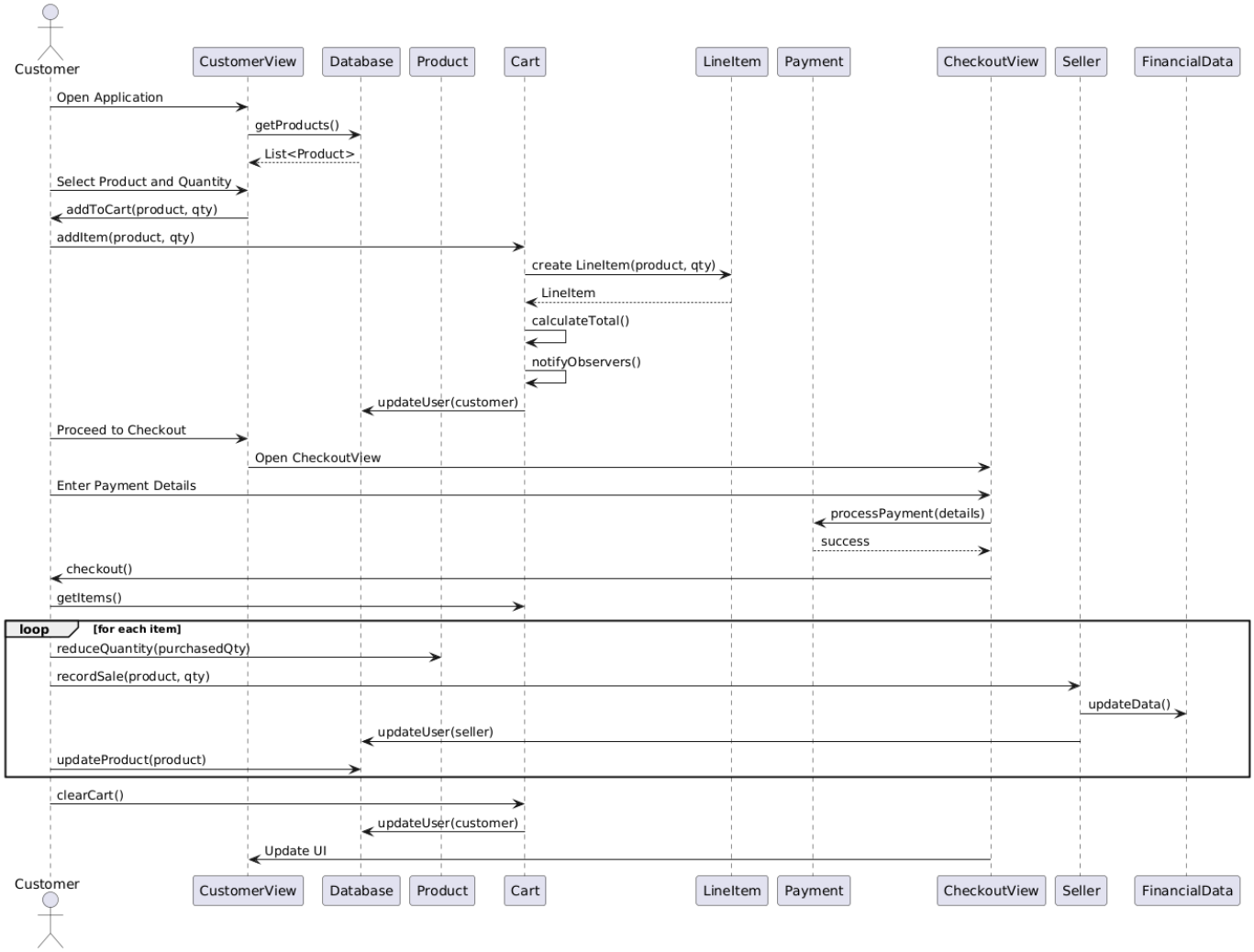
Sequence Diagrams**:**

Customer Logs In



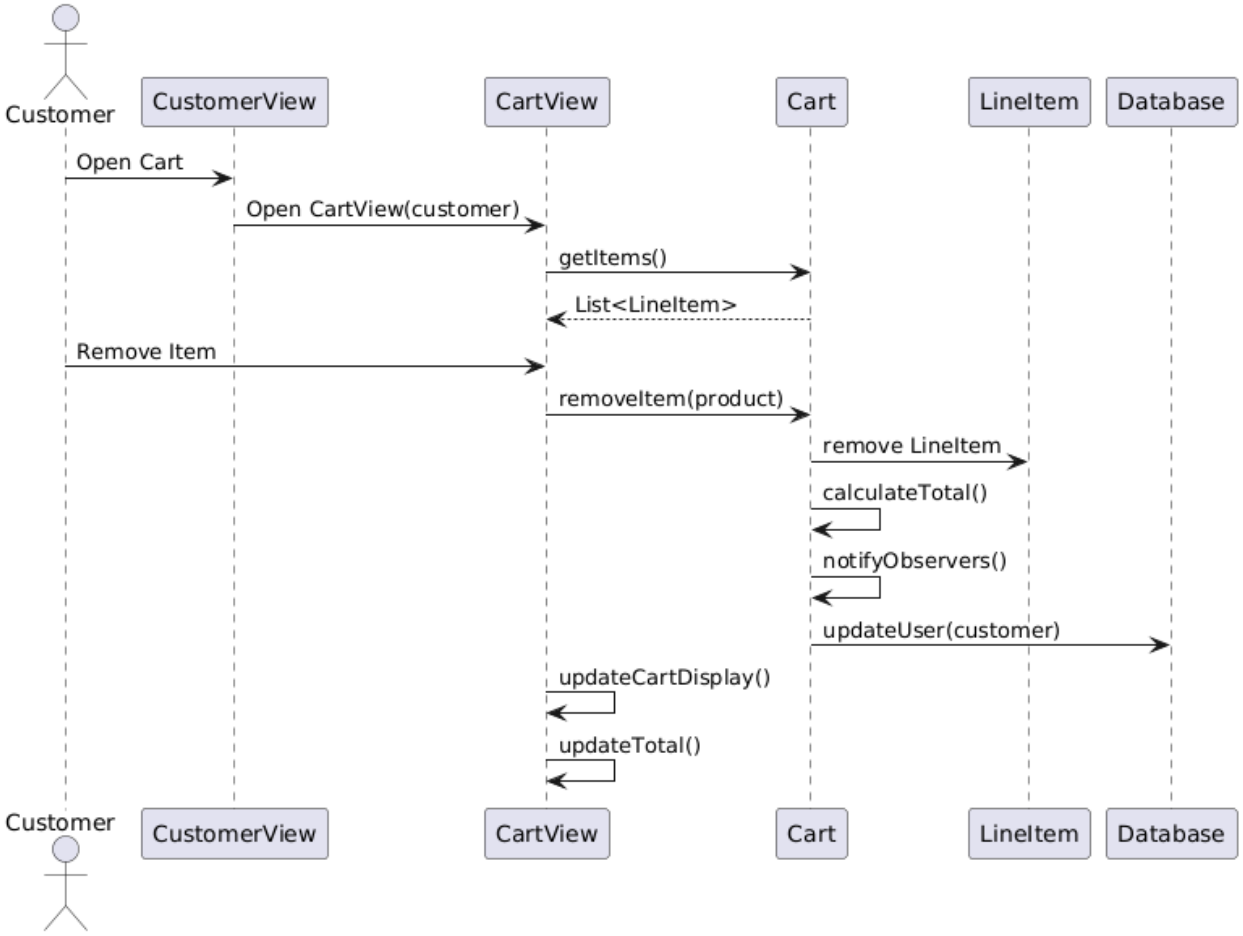
Customer Adds Products to Cart and Checks Out

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



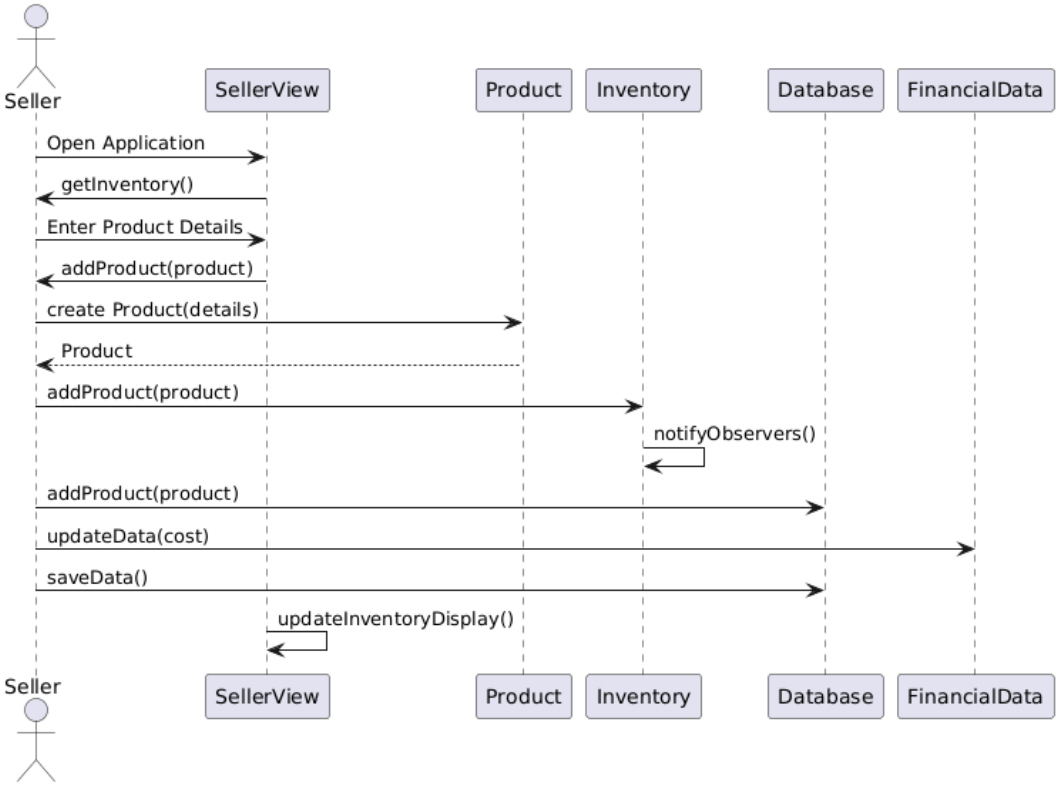
Customer Removes Item from Cart

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



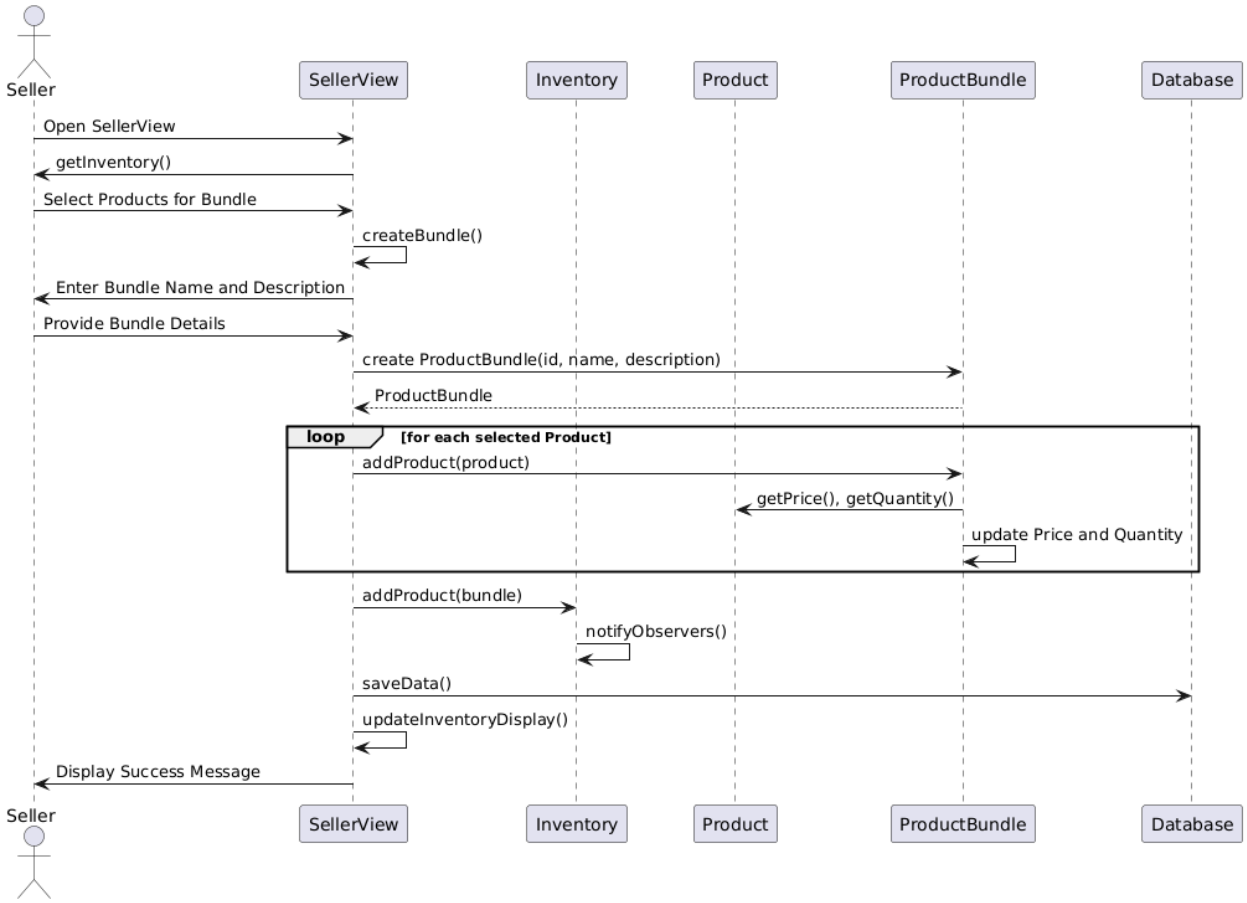
Seller Adds a New Product

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



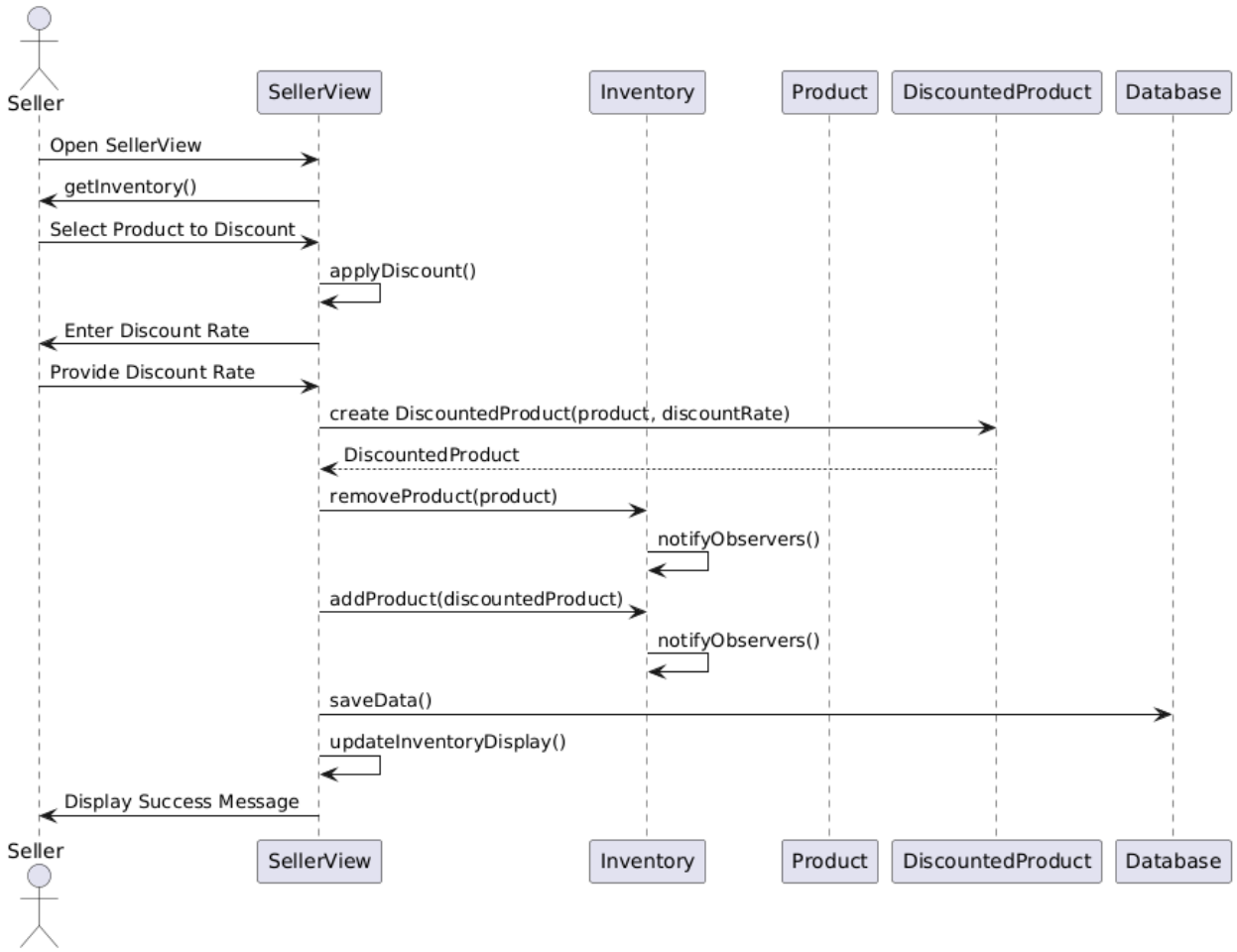
Seller Creates Product Bundle

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



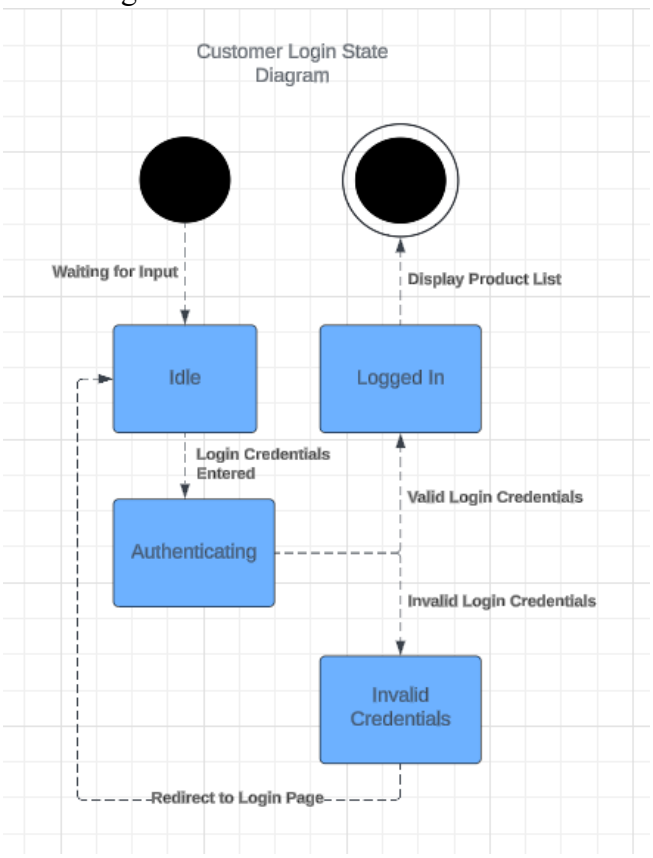
Seller Applies Product Discount

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

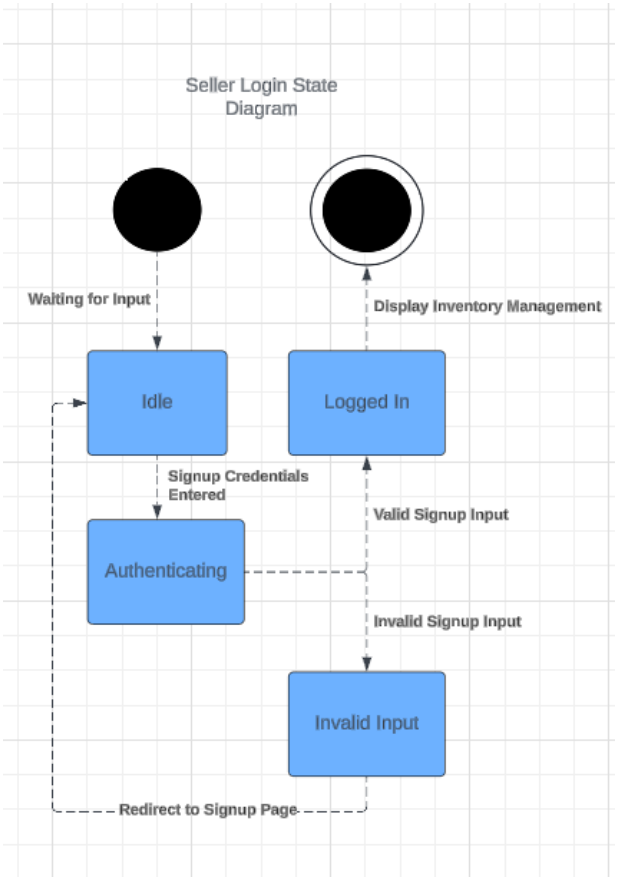


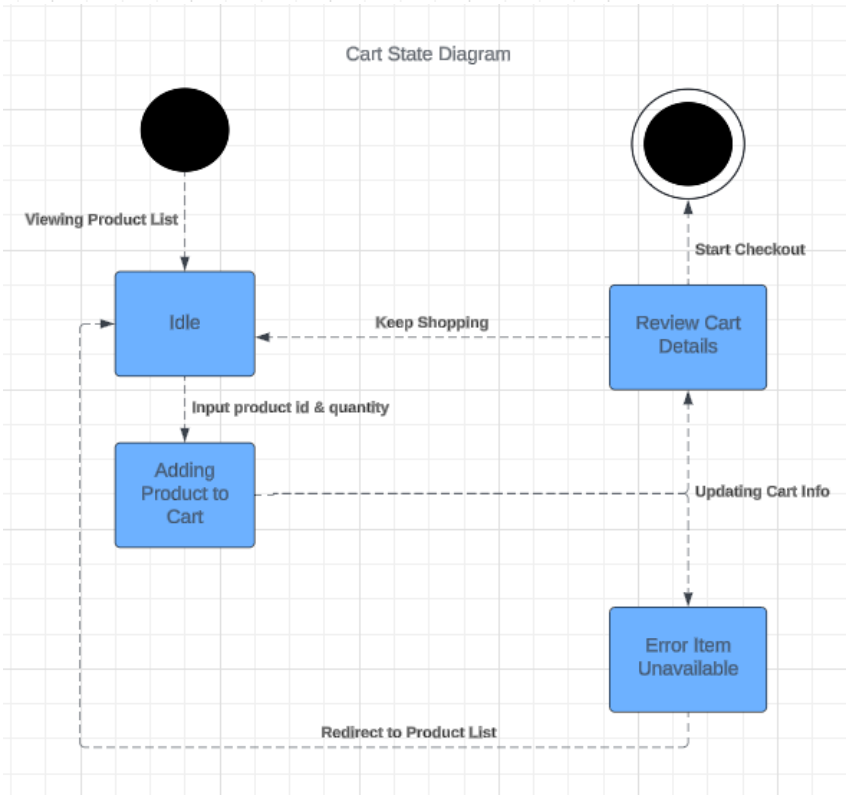
Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

State Diagrams**:**

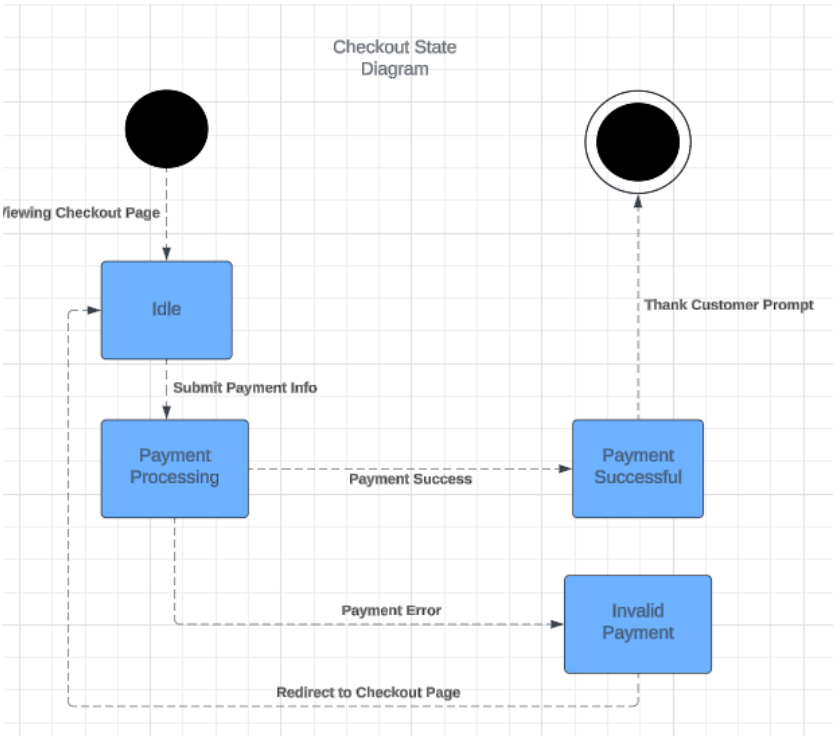


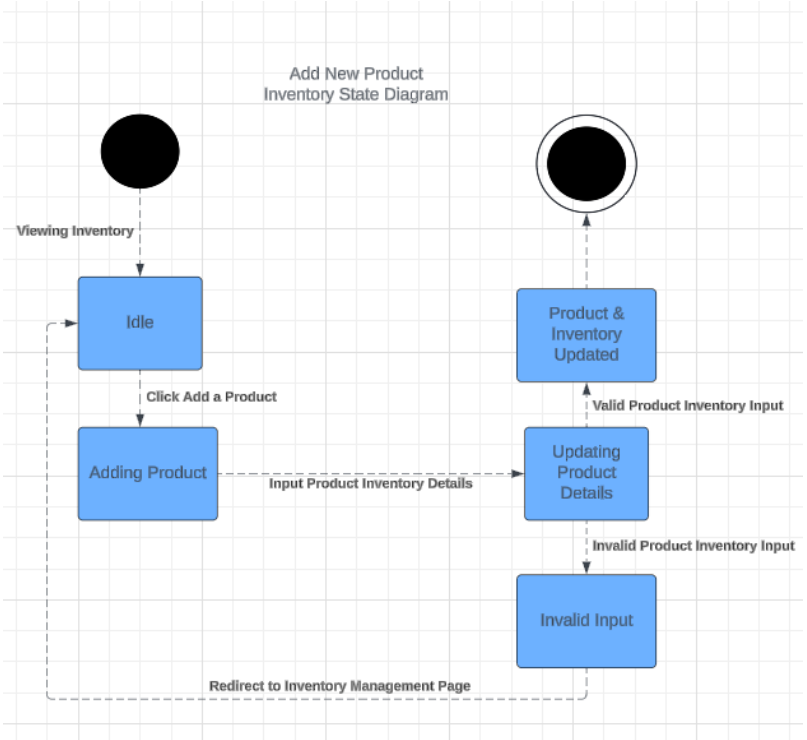
Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



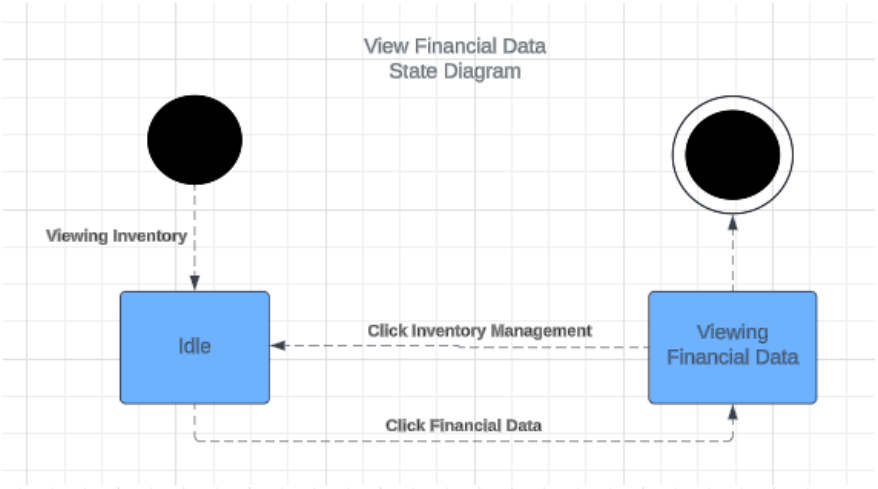


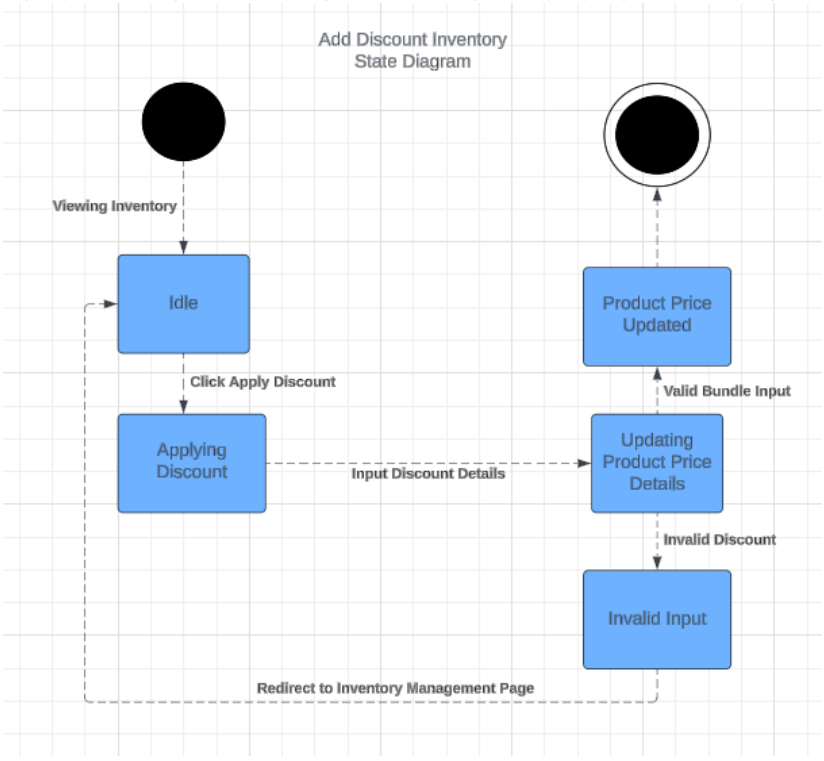
Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



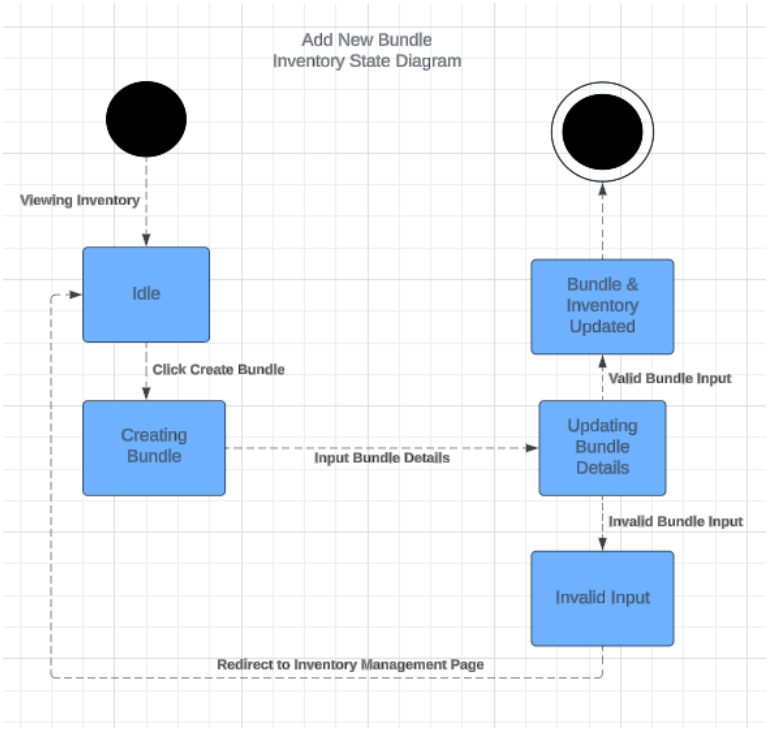


Grace Obeso-Silva
Thomas Allred
Christian Rodrigues





Grace Obeso-Silva
Thomas Allred
Christian Rodrigues



Project Java code:

```
package cop4331.client;

import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

/**
 * The Cart class represents a shopping cart that contains line items and
notifies observers of changes.
 */
public class Cart implements Iterable<LineItem> {
    private List<LineItem> items;
    private transient List<Observer<Cart>> observers;
    private double total;
    private boolean empty;

    /**
     * Constructs an empty Cart.
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     */
    public Cart() {
        this.items = new ArrayList<>();
        this.observers = new ArrayList<>();
        this.total = 0.0;
        this.empty = true;
    }

    /**
     * Returns an unmodifiable list of line items in the cart.
     *
     * @return the unmodifiable list of line items
     */
    public List<LineItem> getItems() {
        return Collections.unmodifiableList(items);
    }

    /**
     * Sets the list of line items in the cart and notifies observers.
     * Also updates the total cost and empty status.
     *
     * @param items the new list of line items
     */
    public void setItems(List<LineItem> items) {
        this.items = new ArrayList<>(items);
        calculateTotal();
        empty = this.items.isEmpty();
        notifyObservers();
    }

    /**
     * Adds a line item to the cart and notifies observers.
     * Also updates the total cost and empty status.
     *
     * @param product the product to add
     * @param qty the quantity of the product
     */
    public void addItem(Product product, int qty) {
        items.add(new LineItem(product, qty));
        calculateTotal();
        empty = items.isEmpty();
        notifyObservers();
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```java
/**
 * Removes a line item from the cart and notifies observers.
 * Also updates the total cost and empty status.
 *
 * @param product the product to remove
 */
public void removeItem(Product product) {
    items.removeIf(item -> item.getProduct().equals(product));
    calculateTotal();
    empty = items.isEmpty();
    notifyObservers();
}
/**
 * Handles quantity updates and notifies observers.
 *
 * @param product
 * @param newQuantity
 */
public void updateItemQuantity(Product product, int newQuantity) {
    for (LineItem item : items) {
        if (item.getProduct().equals(product)) {
            item.setQuantity(newQuantity);
            calculateTotal();
            notifyObservers(); // Notify observers about the change
            return;
        }
    }
}

/**
 * Clears all items from the cart and notifies observers.
 */
public void clearCart() {
    items.clear();
    calculateTotal();
    empty = true;
    notifyObservers();
}

/**
 * Checks if the cart is empty.
 *
 * @return true if the cart is empty, false otherwise
 */
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    public boolean isEmpty() {
        return empty;
    }

    /**
     * Sets the empty status of the cart.
     *
     * @param empty the empty status to set
     */
    public void setEmpty(boolean empty) {
        this.empty = empty;
    }

    /**
     * Returns the total cost of the items in the cart.
     *
     * @return the total cost
     */
    public double getTotal() {
        return total;
    }

    /**
     * Sets the total cost of the cart.
     *
     * @param total the total cost to set
     */
    public void setTotal(double total) {
        this.total = total;
    }

    /**
     * Recalculates the total cost based on the items in the cart.
     * Also updates the empty status.
     */
    public void calculateTotal() {
        total = items.stream()
                        .mapToDouble(item -> item.getProduct().getPrice() *
item.getQuantity())
                        .sum();
        empty = items.isEmpty();
    }

    /**
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * Adds an observer.
     *
     * @param observer the observer to add
     */
    public void addObserver(Observer<Cart> observer) {
        observers.add(observer);
    }

    /**
     * Removes an observer.
     *
     * @param observer the observer to remove
     */
    public void removeObserver(Observer<Cart> observer) {
        observers.remove(observer);
    }

    /**
     * Notifies all observers of changes to the cart.
     */
    public void notifyObservers() {
        for (Observer<Cart> observer : observers) {
            observer.update(this);
        }
    }

    /**
     * Provides an iterator over the line items in the cart.
     *
     * @return an iterator over the line items
     */
    @Override
    public Iterator<LineItem> iterator() {
        return items.iterator();
    }
}

package cop4331.client;

import java.util.List;

/**
 * The Customer class represents a customer user in the system.
 * It extends the User class and includes a shopping cart.
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
 */
public class Customer extends User {
    private Cart cart;

    /**
     * Default constructor.
     * Initializes the customer with an empty cart.
     */
    public Customer() {
        this.cart = new Cart(); // Initialize with an empty cart
    }

    /**
     * Parameterized constructor.
     * Initializes the customer with the specified id, username, and
password, and an empty cart.
     *
     * @param id the customer's id
     * @param username the customer's username
     * @param password the customer's password
     */
    public Customer(String id, String username, String password) {
        super(id, username, password, "customer");
        this.cart = new Cart(); // Initialize with an empty cart
    }

    /**
     * Parameterized constructor.
     * Initializes the customer with the specified id, username, password,
and cart.
     *
     * @param id the customer's id
     * @param username the customer's username
     * @param password the customer's password
     * @param cart the customer's cart
     */
    public Customer(String id, String username, String password, Cart cart) {
        super(id, username, password, "customer");
        this.cart = cart;
    }

    /**
     * Gets the customer's cart.
     *
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * @return the customer's cart
     */
    public Cart getCart() {
        return cart;
    }

    /**
     * Sets the customer's cart.
     *
     * @param cart the new cart
     */
    public void setCart(Cart cart) {
        this.cart = cart;
    }

    /**
     * Displays all available products.
     *
     * @param products the list of products to display
     */
    public void viewProducts(List<Product> products) {
        if (products.isEmpty()) {
            System.out.println("No products available.");
        } else {
            System.out.println("Available Products:");
            for (Product product : products) {
                System.out.println("- " + product.getName() + ": " +
product.getDescription() +
                        " | Price: $" + product.getPrice() + " | Quantity: "
+ product.getQuantity());
            }
        }
    }

    /**
     * Adds a product to the cart.
     *
     * @param product the product to add
     * @param qty the quantity of the product
     */
    public void addToCart(Product product, int qty) {
        if (product.getQuantity() < qty) {
            System.out.println("Insufficient stock for " +
product.getName());
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        } else {
            cart.addItem(product, qty); // Add item to the cart
            // Update the database with the changes
            Database.getInstance().updateUser(this);       // Save updated
customer to users.json
            System.out.println(qty + " x " + product.getName() + " added to
cart.");
        }
    }

    /**
     * Completes the checkout process by clearing the cart and updating
product inventory.
     */
    public void checkout() {
        if (cart.getItems().isEmpty()) {
            System.out.println("Your cart is empty. Add items before checking
out.");
            return;
        }

        for (LineItem item : cart.getItems()) {
            Product product = item.getProduct();
            int purchasedQty = item.getQuantity();

            // Update the product inventory
            product.setQuantity(product.getQuantity() - purchasedQty);


            // Update the seller's financial data
            Seller seller =
Database.getInstance().getSellerById(product.getSellerId());
            if (seller != null) {
                seller.recordSale(product, purchasedQty);
                seller.getInventory().updateProduct(product); // Update
seller's inventory
                Database.getInstance().updateUser(seller); // Save updated
seller to users.json
            }

            Database.getInstance().updateProduct(product); // Save product
changes to products.json
        }
```

```
        // Clear the cart
        cart.clearCart(); // Properly clears the cart
        Database.getInstance().updateUser(this); // Save the cleared cart to
users.json

        System.out.println("Checkout completed successfully. Thank you for
your purchase!");
    }

    /**
     * Logs out the customer.
     */
    @Override
    public void logout() {
        System.out.println("Customer " + username + " logged out.");
    }
}

package cop4331.client;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.JsonSubTypes;
import java.nio.file.Files;
import java.nio.file.Paths;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Singleton class for managing persistent storage of users and products.
 */
public class Database {
    private static Database instance; // Singleton instance
    private List<User> users;         // List of users
    private List<Product> products;   // List of products
    private ObjectMapper objectMapper; // JSON serializer/deserializer

    private static final String USERS_FILE = "users.json";
    private static final String PRODUCTS_FILE = "products.json";
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    // Private constructor for Singleton pattern
    private Database() {
        this.users = new ArrayList<>();
        this.products = new ArrayList<>();
        this.objectMapper = new ObjectMapper();
        loadData();
    }

    /**
     * Returns the Singleton instance of the Database.
     *
     * @return the Database instance.
     */
    public static synchronized Database getInstance() {
        if (instance == null) {
            instance = new Database();
        }
        return instance;
    }

    /**
     * Loads users and products from JSON files into memory.
     */
    public void loadData() {
        try {
            // Debugging users.json
        File usersFile = new File(USERS_FILE);
        if (usersFile.exists()) {
            System.out.println("Loading users.json...");
            System.out.println("JSON Content: " + new
String(Files.readAllBytes(usersFile.toPath())));
            users = objectMapper.readValue(usersFile, new
TypeReference<List<User>>() {});
        } else {
            users = new ArrayList<>();
        }

        // Debugging products.json
        File productsFile = new File(PRODUCTS_FILE);
        if (productsFile.exists()) {
            System.out.println("Loading products.json...");
            System.out.println("JSON Content: " + new
String(Files.readAllBytes(productsFile.toPath())));
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
            products = objectMapper.readValue(productsFile, new
TypeReference<List<Product>>() {});
        } else {
            products = new ArrayList<>();
        }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Adds a product to the database and saves the data.
     *
     * @param product the product to add.
     */
    public void addProduct(Product product) {
        products.add(product);
        saveData(); // Save changes
    }

    /**
     * Saves users and products to their respective JSON files.
     */
    public void saveData() {
        try {
            objectMapper.writeValue(new File(USERS_FILE), users);
            objectMapper.writeValue(new File(PRODUCTS_FILE), products);

            System.out.println("Data saved successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Updates a product in the product list and saves to JSON.
     *
     * @param updatedProduct the product to update.
     */
    public void updateProduct(Product updatedProduct) {
        products.replaceAll(product ->
product.getId().equals(updatedProduct.getId()) ? updatedProduct : product);
        saveData();
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    /**
     * Updates a user in the user list and saves to JSON.
     *
     * @param updatedUser the user to update.
     */
    public void updateUser(User updatedUser) {
        users.replaceAll(user -> user.getId().equals(updatedUser.getId()) ?
updatedUser : user);
        saveData();
    }

    /**
     * Retrieves the list of users.
     *
     * @return the list of users.
     */
    public List<User> getUsers() {
        return users;
    }

    /**
     * Retrieves the list of products.
     *
     * @return the list of products.
     */
    public List<Product> getProducts() {
        return products;
    }

    /**
     * Finds a seller by their unique ID.
     *
     * @param sellerId the unique identifier of the seller.
     * @return the Seller object, or null if not found.
     */
    public Seller getSellerById(String sellerId) {
        for (User user : users) {
            if (user instanceof Seller && user.getId().equals(sellerId)) {
                return (Seller) user;
            }
        }
        return null;
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
}
```

```
package cop4331.client;

/**
 * Represents a discounted product that wraps another product and applies a
discount.
 */
public class DiscountedProduct extends Product {
    private Product product;
    private double discountRate;

    public DiscountedProduct() {
        super();
    }

    public DiscountedProduct(Product product, double discountRate) {
        super(product.getId(), product.getName(), product.getDescription(),
product.getPrice(), product.getQuantity(), product.getSellerId(),
product.getInvoicePrice(), "discountedProduct");
        this.product = product;
        this.discountRate = discountRate;
    }

    @Override
    public double getPrice() {
        return product.getPrice() * (1 - discountRate);
    }

    @Override
    public int getQuantity() {
        return product.getQuantity();
    }

    @Override
    public void setQuantity(int quantity) {
        product.setQuantity(quantity);
    }

    @Override
    public String getName() {
        return product.getName() + " (Discounted)";
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    @Override
    public String getDescription() {
        return product.getDescription() + "\nDiscount: " + (discountRate *
100) + "% off";
    }
}
```

```
package cop4331.client;

import java.io.Serializable;

/**
 * The FinancialData class represents the financial data for a seller,
 * including revenues, costs, and profits.
 */
public class FinancialData implements Serializable {
    private double costs;
    private double revenues;
    private double profits;

    public FinancialData() {
        this.costs = 0.0;
        this.revenues = 0.0;
        this.profits = 0.0;
    }

    public double getCosts() {
        return costs;
    }

    public double getRevenues() {
        return revenues;
    }

    public double getProfits() {
        return profits;
    }

    /**
     * Updates the financial data with the given sale amount and cost amount.
     *
     * @param saleAmount the amount earned from the sale
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * @param costAmount the cost associated with the sale
     */
    public void updateData(double saleAmount, double costAmount) {
        this.revenues += saleAmount;
        this.costs += costAmount;
        calculateProfits();
    }

    private void calculateProfits() {
        this.profits = this.revenues - this.costs;
    }
}


package cop4331.client;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

/**
 * Represents an inventory of products, allowing for management of product
lists
 * and notification of changes to observers.
 */
public class Inventory implements Iterable<Product> {
    /** The list of products in the inventory. */
    private List<Product> products;

    /** The list of observers monitoring the inventory. */
    private transient List<Observer<Inventory>> observers;

    /**
     * Constructs an empty Inventory with no products or observers.
     */
    public Inventory() {
        this.products = new ArrayList<>();
        this.observers = new ArrayList<>();
    }

    /**
     * Retrieves an unmodifiable list of products in the inventory.
     *
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * @return the unmodifiable list of products.
     */
    public List<Product> getProducts() {
        return Collections.unmodifiableList(products);
    }

    /**
     * Sets the list of products in the inventory and notifies all observers
of the change.
     *
     * @param products the new list of products to set.
     */
    public void setProducts(List<Product> products) {
        this.products = new ArrayList<>(products);
        notifyObservers();
    }

    /**
     * Adds a product to the inventory and notifies all observers of the
change.
     *
     * @param product the product to add to the inventory.
     */
    public void addProduct(Product product) {
        products.add(product);
        notifyObservers();
    }

    /**
     * Removes a product from the inventory and notifies all observers of the
change.
     *
     * @param product the product to remove from the inventory.
     */
    public void removeProduct(Product product) {
        products.remove(product);
        notifyObservers();
    }

    /**
     * Finds a product by its ID.
     *
     * @param productId the ID of the product to find.
     * @return the product with the matching ID, or null if not found.
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     */
    public Product findProductById(String productId) {
        for (Product product : products) {
            if (product.getId().equals(productId)) {
                return product;
            }
        }
        return null;
    }

    /**
     * Updates an existing product in the inventory.
     *
     * @param updatedProduct the product with updated information.
     */
    public void updateProduct(Product updatedProduct) {
        for (int i = 0; i < products.size(); i++) {
            if (products.get(i).getId().equals(updatedProduct.getId())) {
                products.set(i, updatedProduct);
                notifyObservers();
                return;
            }
        }
        // Optionally, throw an exception if product not found
    }

    /**
     * Provides an iterator for the products in the inventory.
     *
     * @return an iterator over the products.
     */
    @Override
    public Iterator<Product> iterator() {
        return products.iterator();
    }

    /**
     * Adds an observer to the inventory. Observers are notified whenever the
inventory changes.
     *
     * @param observer the observer to add.
     */
    public void addObserver(Observer<Inventory> observer) {
        observers.add(observer);
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    }

    /**
     * Removes an observer from the inventory.
     *
     * @param observer the observer to remove.
     */
    public void removeObserver(Observer<Inventory> observer) {
        observers.remove(observer);
    }


    /**
     * Notifies all observers of a change to the inventory.
     */
    public void notifyObservers() {
        for (Observer<Inventory> observer : observers) {
            observer.update(this);
        }
    }
}


package cop4331.client;

/**
 * Represents a line item in a cart or inventory, consisting of a product and
its associated quantity.
 */
public class LineItem {
    /** The product associated with this line item. */
    private Product product;

    /** The quantity of the product in this line item. */
    private int quantity;

    /**
     * Default constructor for creating an empty line item.
     */
    public LineItem() {}

    /**
     * Constructs a line item with a specified product and quantity.
     *
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * @param product the product associated with this line item.
     * @param quantity the quantity of the product.
     */
    public LineItem(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }

    /**
     * Retrieves the product associated with this line item.
     *
     * @return the product in this line item.
     */
    public Product getProduct() {
        return product;
    }

    /**
     * Sets the product associated with this line item.
     *
     * @param product the product to set.
     */
    public void setProduct(Product product) {
        this.product = product;
    }

    /**
     * Retrieves the quantity of the product in this line item.
     *
     * @return the quantity of the product.
     */
    public int getQuantity() {
        return quantity;
    }

    /**
     * Sets the quantity of the product in this line item.
     *
     * @param quantity the quantity to set.
     */
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```java
package cop4331.client;

import cop4331.gui.LoginView;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Controller class for the LoginView.
 * Handles authentication and navigation to appropriate views.
 */
public class LoginController {
    private LoginView loginView;
    private Database database;
    private User authenticatedUser;

    /**
     * Constructs a LoginController with the specified LoginView.
     *
     * @param loginView the login view
     */
    public LoginController(LoginView loginView) {
        this.loginView = loginView;
        this.database = Database.getInstance();
        this.authenticatedUser = null;
        initController();
    }

    /**
     * Initializes the controller by adding action listeners.
     */
    private void initController() {
        loginView.getLoginButton().addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                authenticateUser();
            }
        });
    }

    /**
     * Authenticates the user based on entered credentials.
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * Closes login view upon successful login.
     */
    private void authenticateUser() {
        String username = loginView.getUsername();
        String password = loginView.getPassword();

        // Authenticate user
        for (User user : database.getUsers()) {
            if (user.getUsername().equals(username) && user.login(password))
{
                // Successful login
                authenticatedUser = user;
                loginView.dispose();
                return;
            }
        }

        // If authentication fails
        JOptionPane.showMessageDialog(loginView, "Invalid username or
password.", "Login Failed", JOptionPane.ERROR_MESSAGE);
    }

    /**
     * Returns the authenticated user after successful login.
     *
     * @return the authenticated User object, or null if not authenticated
     */
    public User getAuthenticatedUser() {
        return authenticatedUser;
    }
}
```

```
package cop4331.client;

/**
 * The Observer interface should be implemented by any class that wants to be
notified of changes
 * in the observable object.
 *
 * @param <T> the type of the observable object
 */
public interface Observer<T> {
    void update(T observable);
```

```
}
```

```
package cop4331.client;

/**
 * The Payment class handles payment processing.
 * Currently, it simulates payment processing and always returns true.
 */
public class Payment {
    /**
     * Processes the payment with the provided details.
     *
     * @param cardNumber the credit card number
     * @param expiration the expiration date in MM/YY format
     * @param cvv        the CVV code
     * @param amount     the amount to charge
     * @return true if payment is successful, false otherwise
     */
    public boolean processPayment(String cardNumber, String expiration,
String cvv, double amount) {
        // Simulate payment processing logic
        // In a real application, integrate with a payment gateway
        return true;
    }
}
```

```
package cop4331.client;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.JsonSubTypes;

import java.io.Serializable;

@JsonTypeInfo(
    use = JsonTypeInfo.Id.NAME,
    include = JsonTypeInfo.As.EXISTING_PROPERTY, // Change from PROPERTY to
EXISTING_PROPERTY
    property = "type",
    visible = true // Ensures the 'type' property is available to the class
)
@JsonSubTypes({
    @JsonSubTypes.Type(value = Product.class, name = "product"),
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    @JsonSubTypes.Type(value = ProductBundle.class, name = "bundle"),
    @JsonSubTypes.Type(value = DiscountedProduct.class, name =
"discountedProduct")
})
/**
 * Represents a product in the system, including its details such as ID,
name, description, price, and quantity.
 */
public class Product implements Serializable {
    /** The unique identifier of the product. */
    private String id;

    /** The name of the product. */
    private String name;

    /** A brief description of the product. */
    private String description;

    /** The price of the product. */
    private double price;

    /** The quantity of the product available in stock. */
    private int quantity;

    /** The unique identifier of the seller of the product. */
    private String sellerId;

    /** The invoice price of the product. */
    private double invoicePrice;

    private String type;

    /**
     * Default constructor for creating an empty product.
     */
    public Product() {}

    /**
     * Constructs a product with specified details.
     *
     * @param id the unique identifier of the product.
     * @param name the name of the product.
     * @param description a brief description of the product.
     * @param price the price of the product.
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * @param quantity the quantity of the product available in stock.
     * @param sellerId sellerId the unique identifier of the seller of the
product.
     * @param invoicePrice the invoice price of the product.
     */
    public Product(String id, String name, String description, double price,
int quantity, String sellerId, double invoicePrice, String type) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
        this.quantity = quantity;
        this.sellerId = sellerId;
        this.invoicePrice = invoicePrice;
        this.type = type;
    }

    /**
     * Retrieves the unique identifier of the product.
     *
     * @return the product ID.
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the unique identifier of the product.
     *
     * @param id the product ID to set.
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * Retrieves the name of the product.
     *
     * @return the product name.
     */
    public String getName() {
        return name;
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```java
    /**
     * Retrieves the type of the product.
     *
     * @return the type of the product.
     */
    public String getType() {
        return type;
    }

    /**
     * Sets the type of the product.
     *
     * @param type the type of the product.
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     * Sets the name of the product.
     *
     * @param name the product name to set.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Retrieves the description of the product.
     *
     * @return the product description.
     */
    public String getDescription() {
        return description;
    }

    /**
     * Sets the description of the product.
     *
     * @param description the product description to set.
     */
    public void setDescription(String description) {
        this.description = description;
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```java
    /**
     * Retrieves the price of the product.
     *
     * @return the product price.
     */
    public double getPrice() {
        return price;
    }

    /**
     * Sets the price of the product.
     *
     * @param price the product price to set.
     */
    public void setPrice(double price) {
        this.price = price;
    }

    /**
     * Retrieves the quantity of the product available in stock.
     *
     * @return the product quantity.
     */
    public int getQuantity() {
        return quantity;
    }

    /**
     * Sets the quantity of the product available in stock.
     *
     * @param quantity the product quantity to set.
     */
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    /**
     * Retrieves the seller ID of the product.
     *
     * @return the seller ID.
     */
    public String getSellerId() {
        return sellerId;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    }

    /**
     * Sets the seller ID of the product.
     *
     * @param sellerId the seller ID to set.
     */
    public void setSellerId(String sellerId) {
        this.sellerId = sellerId;
    }

    /**
     * Retrieves the invoice price of the product.
     *
     * @return the invoice price.
     */
    public double getInvoicePrice() {
        return invoicePrice;
    }

    /**
     * Sets the invoice price of the product.
     *
     * @param invoicePrice the invoice price to set.
     */
    public void setInvoicePrice(double invoicePrice) {
        this.invoicePrice = invoicePrice;
    }
}


package cop4331.client;

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

/**
 * Represents a product bundle that consists of multiple products.
 * Applies a 10% discount to the total price of the bundled products.
 */
public class ProductBundle extends Product {
    private List<Product> products;

    public ProductBundle() {
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        super();
        this.products = new ArrayList<>();
    }

    public ProductBundle(String id, String name, String description) {
        super(id, name, description, 0.0, 0, "", 0.0, "bundle");
        this.products = new ArrayList<>();
    }

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

    public void addProduct(Product product) {
        this.products.add(product);
        // Update sellerId if necessary
        if (this.getSellerId() == null || this.getSellerId().isEmpty()) {
            this.setSellerId(product.getSellerId());
        }
        // Recalculate price and quantity
        this.setPrice(getPrice());
        this.setQuantity(getQuantity());
    }

    @Override
    public double getPrice() {
        double totalPrice =
products.stream().mapToDouble(Product::getPrice).sum();
        return totalPrice * 0.9; // Apply 10% discount
    }

    @Override
    public int getQuantity() {
        return
products.stream().mapToInt(Product::getQuantity).min().orElse(0);
    }

    @Override
    public void setQuantity(int quantity) {
        for (Product product : products) {
            product.setQuantity(quantity);
        }
    }
```

```
    @Override
    public String getDescription() {
        StringBuilder description = new StringBuilder(super.getDescription()
+ "\nIncludes:\n");
        for (Product product : products) {
            description.append("- ").append(product.getName()).append("\n");
        }
        return description.toString();
    }
}
```

```
package cop4331.client;

import java.io.Serializable;

/**
 * Represents a seller in the system, managing an inventory of products.
 * Inherits from the {@link User} class.
 */
public class Seller extends User implements Serializable {
    private Inventory inventory;
    private FinancialData financialData;

    /**
     * Default constructor that initializes the seller with an empty
inventory.
     */
    public Seller() {
        super();
        this.inventory = new Inventory();
        this.financialData = new FinancialData();
    }

    /**
     * Constructs a seller with the specified ID, username, and password,
     * and initializes an empty inventory.
     *
     * @param id the unique identifier for the seller.
     * @param username the seller's username.
     * @param password the seller's password.
     */
    public Seller(String id, String username, String password) {
        super(id, username, password, "seller");
        this.inventory = new Inventory();
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        this.financialData = new FinancialData();
    }

    /**
     * Retrieves the seller's inventory.
     *
     * @return the inventory managed by the seller.
     */
    public Inventory getInventory() {
        return inventory;
    }

    /**
     * Sets the seller's inventory.
     *
     * @param inventory the inventory to set.
     */
    public void setInventory(Inventory inventory) {
        this.inventory = inventory;
    }

    /**
     * Adds a product to the seller's inventory and updates financial data.
     *
     * @param product the product to add to the inventory.
     */
    public void addProduct(Product product) {
        product.setSellerId(this.getId()); // Set the seller reference
        inventory.addProduct(product);

        // Add the product to the global products list in the Database
        Database.getInstance().addProduct(product);

        // Update financial data with the cost of the product
        double costAmount = product.getInvoicePrice() *
product.getQuantity();
        financialData.updateData(0, costAmount); // No revenue yet, only cost
        System.out.println("Product added to inventory: " +
product.getName());
    }

    /**
     * Updates the inventory and notifies all observers of any changes.
     */
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    public void updateInventory() {
        inventory.notifyObservers();
        System.out.println("Inventory updated.");
    }

    /**
     * Views the financial data, displaying revenues, costs, and profits.
     */
    public void viewFinancialData() {
        double revenues = financialData.getRevenues();
        double costs = financialData.getCosts();
        double profits = financialData.getProfits();

        System.out.println("Financial Data:");
        System.out.println("Total Revenues: $" + String.format("%.2f",
revenues));
        System.out.println("Total Costs: $" + String.format("%.2f", costs));
        System.out.println("Total Profits: $" + String.format("%.2f",
profits));
    }

    /**
     * Records a sale of a product and updates the financial data.
     *
     * @param product the product sold
     * @param quantity the quantity sold
     */
    public void recordSale(Product product, int quantity) {
        double saleAmount = product.getPrice() * quantity;
        double costAmount = product.getInvoicePrice() * quantity; // Include
cost
        financialData.updateData(saleAmount, costAmount);
    }

    /**
     * Gets the seller's financial data.
     *
     * @return the seller's financial data
     */
    public FinancialData getFinancialData() {
        return financialData;
    }

    /**
```

```
     * Logs out the seller from the system and displays a logout message.
     */
    @Override
    public void logout() {
        System.out.println("Seller logged out.");
    }
}


package cop4331.client;

import cop4331.gui.*;
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

/**
 * Entry point of the application.
 * Initializes and starts the shopping cart application.
 */
public class SystemApp {

    /**
     * Main method to start the application.
     *
     * @param args command-line arguments
     */
    public static void main(String[] args) {
        SystemApp app = new SystemApp();
        app.initialize();
        app.startApplication();
    }

    /**
     * Initializes the application.
     * Loads data and performs any necessary setup.
     */
    private void initialize() {
        // Initialize the database and load data
        Database db = Database.getInstance();

    }

    /**
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     * Starts the application by showing the login view and handling
navigation.
     */
    private void startApplication() {
        // Ensure GUI creation is done on the Event Dispatch Thread
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                // Show login view
                LoginView loginView = new LoginView();
                LoginController loginController = new
LoginController(loginView);
                loginView.setVisible(true);

                // Wait for the login view to be disposed
                loginView.addWindowListener(new
java.awt.event.WindowAdapter() {
                    @Override
                    public void windowClosed(java.awt.event.WindowEvent
windowEvent) {
                        User authenticatedUser =
loginController.getAuthenticatedUser();
                        if (authenticatedUser != null) {
                            openUserView(authenticatedUser);
                        } else {
                            // Exit the application if authentication failed
or window was closed
                            System.exit(0);
                        }
                    }
                });
            }
        });
    }

    /**
     * Opens the appropriate view based on the user type.
     *
     * @param user the authenticated User object
     */
    private void openUserView(User user) {
        if (user instanceof Customer) {
            // Open Customer View
            CustomerView customerView = new CustomerView((Customer) user);
```

```
            customerView.setVisible(true);
        } else if (user instanceof Seller) {
            // Open Seller View
            SellerView sellerView = new SellerView((Seller) user);
            sellerView.setVisible(true);
        } else {
            // Handle other user types if necessary
            JOptionPane.showMessageDialog(null, "Unknown user type.",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

```
package cop4331.client;

import com.fasterxml.jackson.annotation.JsonSubTypes;
import com.fasterxml.jackson.annotation.JsonTypeInfo;

/**
 * Abstract base class representing a user in the system.
 * A user can be a {@link Customer} or a {@link Seller}, with unique
credentials.
 * Provides common properties and methods for all user types.
 */
@JsonTypeInfo(
    use = JsonTypeInfo.Id.NAME,
    include = JsonTypeInfo.As.PROPERTY,
    property = "type",
    visible = true
)
@JsonSubTypes({
    @JsonSubTypes.Type(value = Customer.class, name = "customer"),
    @JsonSubTypes.Type(value = Seller.class, name = "seller")
})
public abstract class User {
    /** The unique identifier for the user. */
    protected String id;

    /** The username for the user. */
    protected String username;

    /** The password for the user. */
    protected String password;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    /**
     * The type of the user.
     */
    protected String type;

    /**
     * Default constructor required for Jackson deserialization.
     */
    public User() {}

    /**
     * Constructs a user with the specified ID, username, and password.
     *
     * @param id the unique identifier for the user.
     * @param username the username for the user.
     * @param password the password for the user.
     */
    public User(String id, String username, String password, String type) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.type = type;
    }

    /**
     * Authenticates the user by comparing the provided password with the
stored password.
     *
     * @param password the password to authenticate.
     * @return true if the provided password matches the stored password,
false otherwise.
     */
    public boolean login(String password) {
        return this.password.equals(password);
    }

    /**
     * Logs out the user and displays a logout message.
     */
    public void logout() {
        System.out.println(username + " logged out.");
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    /**
     * Retrieves the type of the user.
     *
     * @return the type of the user.
     */
    public String getType() {
        return type;
    }

    /**
     * Sets the type of the user.
     *
     * @param type the type of the user.
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     * Retrieves the unique identifier for the user.
     *
     * @return the user ID.
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the unique identifier for the user.
     *
     * @param id the user ID to set.
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * Retrieves the username for the user.
     *
     * @return the username.
     */
    public String getUsername() {
        return username;
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    /**
     * Sets the username for the user.
     *
     * @param username the username to set.
     */
    public void setUsername(String username) {
        this.username = username;
    }

    /**
     * Retrieves the password for the user.
     *
     * @return the password.
     */
    public String getPassword() {
        return password;
    }

    /**
     * Sets the password for the user.
     *
     * @param password the password to set.
     */
    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
package cop4331.gui;

import cop4331.client.Cart;
import cop4331.client.Customer;
import cop4331.client.Database;
import cop4331.client.LineItem;
import cop4331.client.Product;
import cop4331.gui.CheckoutView;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;
import java.awt.*;
import java.util.List;

public class CartView extends JFrame {
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    private Customer customer;
    private Cart cart;

    private JTable cartTable;
    private CartTableModel cartTableModel;

    private JLabel totalLabel;

    private JButton updateCartButton;
    private JButton checkoutButton;

    public CartView(Customer customer) {
        this.customer = customer;
        this.cart = customer.getCart();

        setTitle("Your Cart");
        setSize(600, 400);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        initializeComponents();
        layoutComponents();
        registerListeners();

        updateTotal();
    }

    private void initializeComponents() {
        // Initialize the cart table model and table
        cartTableModel = new CartTableModel(cart);
        cartTable = new JTable(cartTableModel);

        // Allow editing of the Quantity column
        cartTable.getColumnModel().getColumn(2).setCellEditor(new
DefaultCellEditor(new JTextField()));

        // Set the Remove column to display buttons
        cartTable.getColumnModel().getColumn(4).setCellRenderer(new
ButtonRenderer());
        cartTable.getColumnModel().getColumn(4).setCellEditor(new
ButtonEditor(new JCheckBox()));

        totalLabel = new JLabel("Total: $0.00");
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        updateCartButton = new JButton("Update Cart");
        checkoutButton = new JButton("Proceed to Checkout");
    }

    private void layoutComponents() {
        JPanel mainPanel = new JPanel(new BorderLayout());

        // Cart items table
        JScrollPane tableScrollPane = new JScrollPane(cartTable);
        mainPanel.add(tableScrollPane, BorderLayout.CENTER);

        // Bottom panel with total and buttons
        JPanel bottomPanel = new JPanel(new BorderLayout());

        // Total label
        bottomPanel.add(totalLabel, BorderLayout.WEST);

        // Buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(updateCartButton);
        buttonPanel.add(checkoutButton);
        bottomPanel.add(buttonPanel, BorderLayout.EAST);

        mainPanel.add(bottomPanel, BorderLayout.SOUTH);

        setContentPane(mainPanel);
    }

    private void registerListeners() {
        updateCartButton.addActionListener(e -> updateCart());
        checkoutButton.addActionListener(e -> proceedToCheckout());
    }

    private void updateCart() {
        // Stop cell editing to ensure changes are committed
        if (cartTable.isEditing()) {
            cartTable.getCellEditor().stopCellEditing();
        }

        // Apply changes to quantities
        cartTableModel.applyChanges();

        // Update the total
        updateTotal();
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        // Save the updated cart to the database
        Database.getInstance().updateUser(customer);
    }

    private void updateTotal() {
        totalLabel.setText("Total: $" + String.format("%.2f",
cart.getTotal()));
    }

    private void proceedToCheckout() {
        if (cart.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Your cart is empty!",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Open the CheckoutView
        CheckoutView checkoutView = new CheckoutView(this, customer);
        checkoutView.setVisible(true);

        // After checkout, refresh the cart view
        cartTableModel.fireTableDataChanged();
        updateTotal();

        // Close CartView if the cart is empty after checkout
        if (cart.isEmpty()) {
            dispose();
        }
    }

    // Inner class for the cart table model
    private class CartTableModel extends AbstractTableModel {
        private final String[] columnNames = {"Product Name", "Price per
Unit", "Quantity", "Total Price", "Remove"};
        private Cart cart;

        public CartTableModel(Cart cart) {
            this.cart = cart;
        }

        @Override
        public int getColumnCount() {
            return columnNames.length;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        }

        @Override
        public String getColumnName(int column) {
            return columnNames[column];
        }

        @Override
        public int getRowCount() {
            return cart.getItems().size();
        }

        @Override
        public Object getValueAt(int row, int col) {
            LineItem item = cart.getItems().get(row);
            Product product = item.getProduct();
            switch (col) {
                case 0:
                    return product.getName();
                case 1:
                    return "$" + String.format("%.2f", product.getPrice());
                case 2:
                    return item.getQuantity();
                case 3:
                    return "$" + String.format("%.2f", product.getPrice() *
item.getQuantity());
                case 4:
                    return "Remove";
                default:
                    return null;
            }
        }

        @Override
        public boolean isCellEditable(int row, int col) {
            // Allow editing of Quantity column and Remove button
            return col == 2 || col == 4;
        }

        @Override
        public void setValueAt(Object value, int row, int col) {
            LineItem item = cart.getItems().get(row);
            if (col == 2) {
                try {
```

```
                    int newQuantity = Integer.parseInt(value.toString());
                    if (newQuantity > 0) {
                        cart.updateItemQuantity(item.getProduct(),
newQuantity);
                        fireTableCellUpdated(row, col);
                        fireTableCellUpdated(row, 3); // Update the total
price column
                        updateTotal();
                    } else {
                        JOptionPane.showMessageDialog(CartView.this,
"Quantity must be greater than 0.", "Invalid Quantity",
JOptionPane.ERROR_MESSAGE);
                    }
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(CartView.this, "Invalid
quantity.", "Error", JOptionPane.ERROR_MESSAGE);
                }
                Database.getInstance().updateUser(customer); // Save changes
            } else if (col == 4) {
                // Remove item
                cart.removeItem(item.getProduct());
                fireTableDataChanged();
                updateTotal();
                Database.getInstance().updateUser(customer); // Save changes
            }
        }

        public void applyChanges() {
            // Recalculate the total in the cart
            cart.calculateTotal();
            // Refresh the table
            fireTableDataChanged();
        }
    }

    // Renderer and editor for the Remove button in the table
    private class ButtonRenderer extends JButton implements
javax.swing.table.TableCellRenderer {
        public ButtonRenderer() {
            setText("Remove");
        }

        @Override
```

```
        public Component getTableCellRendererComponent(JTable table, Object
value,
                                                       boolean isSelected,
boolean hasFocus, int row, int column) {
            return this;
        }
    }

    private class ButtonEditor extends DefaultCellEditor {
        private JButton button;
        private LineItem item;

        public ButtonEditor(JCheckBox checkBox) {
            super(checkBox);
            button = new JButton("Remove");
            button.addActionListener(e -> {
                fireEditingStopped();
                cart.removeItem(item.getProduct());
                cartTableModel.fireTableDataChanged();
                updateTotal();
                Database.getInstance().updateUser(customer); // Save changes
            });
        }

        @Override
        public Component getTableCellEditorComponent(JTable table, Object
value,
                                                     boolean isSelected, int
row, int column) {
            this.item = cart.getItems().get(row); // Capture the LineItem
before any changes
            return button;
        }

        @Override
        public Object getCellEditorValue() {
            return "Remove";
        }
    }
}
```

```
package cop4331.gui;

import cop4331.client.Customer;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```java
import cop4331.client.Cart;
import cop4331.client.Payment;

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

/**
 * The CheckoutView class provides a window where customers can input credit card information to complete the checkout process.
 */
public class CheckoutView extends JDialog {
    private Customer customer;
    private Cart cart;

    private JTextField cardNumberField;
    private JTextField expirationField;
    private JTextField cvvField;
    private JButton submitButton;
    private JButton cancelButton;
    private JLabel totalLabel;

    public CheckoutView(JFrame parent, Customer customer) {
        super(parent, "Checkout", true);
        this.customer = customer;
        this.cart = customer.getCart();

        initializeComponents();
        layoutComponents();
        registerListeners();

        setSize(400, 250);
        setLocationRelativeTo(parent);
    }

    private void initializeComponents() {
        cardNumberField = new JTextField(16);
        expirationField = new JTextField(5);
        cvvField = new JTextField(3);
        submitButton = new JButton("Submit Payment");
        cancelButton = new JButton("Cancel");
        totalLabel = new JLabel("Total: $" + String.format("%.2f", cart.getTotal()));
    }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    private void layoutComponents() {
        JPanel mainPanel = new JPanel(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        gbc.insets = new Insets(10, 10, 5, 10);
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.anchor = GridBagConstraints.EAST;
        mainPanel.add(new JLabel("Card Number:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        mainPanel.add(cardNumberField, gbc);

        gbc.gridx = 0;
        gbc.gridy++;
        gbc.fill = GridBagConstraints.NONE;
        mainPanel.add(new JLabel("Expiration (MM/YY):"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        mainPanel.add(expirationField, gbc);

        gbc.gridx = 0;
        gbc.gridy++;
        gbc.fill = GridBagConstraints.NONE;
        mainPanel.add(new JLabel("CVV:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        mainPanel.add(cvvField, gbc);

        gbc.gridx = 0;
        gbc.gridy++;
        gbc.fill = GridBagConstraints.NONE;
        mainPanel.add(new JLabel(""), gbc); // Spacer

        gbc.gridx = 1;
        mainPanel.add(totalLabel, gbc);

        gbc.gridx = 0;
        gbc.gridy++;
        gbc.gridwidth = 2;
        gbc.anchor = GridBagConstraints.CENTER;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(submitButton);
        buttonPanel.add(cancelButton);
        mainPanel.add(buttonPanel, gbc);

        setContentPane(mainPanel);
    }

    private void registerListeners() {
        submitButton.addActionListener(e -> processPayment());
        cancelButton.addActionListener(e -> dispose());
    }

    private void processPayment() {
        String cardNumber = cardNumberField.getText().trim();
        String expiration = expirationField.getText().trim();
        String cvv = cvvField.getText().trim();
        double amount = cart.getTotal();

        // Basic validation
        if (cardNumber.isEmpty() || expiration.isEmpty() || cvv.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please fill in all fields.",
"Missing Information", JOptionPane.ERROR_MESSAGE);
            return;
        }

        if (!cardNumber.matches("\\d{16}")) {
            JOptionPane.showMessageDialog(this, "Invalid card number. Must be
16 digits.", "Invalid Card Number", JOptionPane.ERROR_MESSAGE);
            return;
        }

        if (!expiration.matches("(0[1-9]|1[0-2])/\\d{2}")) {
            JOptionPane.showMessageDialog(this, "Invalid expiration date.
Format MM/YY.", "Invalid Expiration Date", JOptionPane.ERROR_MESSAGE);
            return;
        }

        if (!cvv.matches("\\d{3}")) {
            JOptionPane.showMessageDialog(this, "Invalid CVV. Must be 3
digits.", "Invalid CVV", JOptionPane.ERROR_MESSAGE);
            return;
        }
```

```
        // Simulate payment processing
        Payment payment = new Payment();
        boolean paymentSuccess = payment.processPayment(cardNumber,
expiration, cvv, amount);
        if (paymentSuccess) {
            // Clear the cart
            customer.checkout(); // Reduces stock quantity

            // Inform the user
            JOptionPane.showMessageDialog(this, "Payment successful! Thank
you for your purchase.");

            // Close the checkout window
            dispose();
        } else {
            JOptionPane.showMessageDialog(this, "Payment failed. Please try
again.", "Payment Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}


package cop4331.gui;

import cop4331.client.Customer;
import cop4331.client.Database;
import cop4331.client.Product;
import cop4331.client.Cart;
import cop4331.client.Observer;
import cop4331.client.LineItem;

import cop4331.gui.CheckoutView; // Import CheckoutView

import javax.swing.*;
import java.awt.*;
import java.util.List;

/**
 * GUI class representing the customer view in the system.
 * Provides functionality for customers to browse products, add items to
their cart,
 * view their cart, and proceed to checkout.
 */
public class CustomerView extends JFrame implements Observer<Cart> {
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    /** The customer using this view. */
    private Customer customer;

    /** Text area for displaying available products. */
    private JTextArea productDisplay;

    /** Text area for displaying items in the customer's cart. */
    private JTextArea cartDisplay;

    /** Text field for entering the ID of the product to add to the cart. */
    private JTextField productIdField;

    /** Text field for entering the quantity of the product to add to the
cart. */
    private JTextField quantityField;

    /** Button for adding a product to the cart. */
    private JButton addToCartButton;

    /** Button for proceeding to checkout. */
    private JButton checkoutButton;

    /** Button displaying the cart icon with the total number of items in the
cart. */
    private JButton cartIcon;

    /**
     * Constructs the customer view with the specified customer.
     *
     * @param customer the customer using this view.
     */
    public CustomerView(Customer customer) {
        this.customer = customer;
        setTitle("Customer View - " + customer.getUsername());
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Product Display
        productDisplay = new JTextArea();
        productDisplay.setEditable(false);
        JScrollPane productScroll = new JScrollPane(productDisplay);
        productScroll.setBorder(BorderFactory.createTitledBorder("Available
Products"));
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        add(productScroll, BorderLayout.WEST);

        // Cart Display
        cartDisplay = new JTextArea();
        cartDisplay.setEditable(false);
        JScrollPane cartScroll = new JScrollPane(cartDisplay);
        cartScroll.setBorder(BorderFactory.createTitledBorder("Your Cart"));
        add(cartScroll, BorderLayout.EAST);

        // Bottom Panel
        JPanel bottomPanel = new JPanel(new GridLayout(3, 2, 5, 5));
        bottomPanel.add(new JLabel("Product ID:"));
        productIdField = new JTextField();
        bottomPanel.add(productIdField);

        bottomPanel.add(new JLabel("Quantity:"));
        quantityField = new JTextField();
        bottomPanel.add(quantityField);

        addToCartButton = new JButton("Add to Cart");
        checkoutButton = new JButton("Checkout");
        bottomPanel.add(addToCartButton);
        bottomPanel.add(checkoutButton);

        add(bottomPanel, BorderLayout.SOUTH);

        // Add cart icon to the top right
        JPanel topPanel = new JPanel(new BorderLayout());
        cartIcon = new JButton("Cart (0)");
        topPanel.add(cartIcon, BorderLayout.EAST);
        add(topPanel, BorderLayout.NORTH);

        // Register as an observer of the cart
        customer.getCart().addObserver(this);

        // Add ActionListener to cartIcon
        cartIcon.addActionListener(e -> {
            CartView cartView = new CartView(customer);
            cartView.setVisible(true);
        });

        setLocationRelativeTo(null);

        // Load products into the display
```

```
        updateProductDisplay(Database.getInstance().getProducts());

        // Initial update of cart display and icon
        updateCartDisplay();
        updateCartIcon();

        // Add listeners
        addToCartButton.addActionListener(e -> {
            try {
                String productId = getProductId();
                int qty = getQuantity();

                // Find the product in the database
                Product product =
Database.getInstance().getProducts().stream()
                        .filter(p -> p.getId().equals(productId))
                        .findFirst()
                        .orElse(null);

                if (product == null) {
                    JOptionPane.showMessageDialog(this, "Product not found!",
"Error", JOptionPane.ERROR_MESSAGE);
                } else if (product.getQuantity() < qty) {
                    JOptionPane.showMessageDialog(this, "Insufficient stock
for " + product.getName(), "Error", JOptionPane.ERROR_MESSAGE);
                } else {
                    customer.addToCart(product, qty);
                    // No need to manually update cart display and icon here
since the observer pattern handles it
                    JOptionPane.showMessageDialog(this, qty + " x " +
product.getName() + " added to cart!");
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this, "Invalid quantity!",
"Error", JOptionPane.ERROR_MESSAGE);
            }
        });

        checkoutButton.addActionListener(e -> {
            if (customer.getCart().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Your cart is empty!",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
            // Open the CheckoutView
            CheckoutView checkoutView = new CheckoutView(this, customer);
            checkoutView.setVisible(true);

            // After checkout, refresh the UI
            updateCartDisplay();
            updateCartIcon();
        });
    }

    /**
     * Updates the product display with the available products.
     *
     * @param products the list of products to display.
     */
    public void updateProductDisplay(List<Product> products) {
        productDisplay.setText("");
        for (Product product : products) {
            productDisplay.append(product.getId() + ": " + product.getName()
+
                    " - $" + product.getPrice() + " (Stock: " +
product.getQuantity() + ")\n");
        }
    }

    /**
     * Updates the cart display with the items currently in the customer's
cart.
     */
    public void updateCartDisplay() {
        cartDisplay.setText("");
        customer.getCart().getItems().forEach(item -> cartDisplay.append(
                item.getProduct().getName() + " x " + item.getQuantity() +
"\n"));
    }

    /**
     * Updates the cart icon to display the total number of items in the
customer's cart.
     */
    public void updateCartIcon() {
        int itemCount = customer.getCart().getItems().stream()
                .mapToInt(LineItem::getQuantity)
```

```
                .sum();
        cartIcon.setText("Cart (" + itemCount + ")");
    }

    /**
     * Retrieves the product ID entered by the customer.
     *
     * @return the entered product ID.
     */
    public String getProductId() {
        return productIdField.getText();
    }

    /**
     * Retrieves the quantity of the product entered by the customer.
     *
     * @return the entered quantity.
     * @throws NumberFormatException if the entered quantity is not a valid
integer.
     */
    public int getQuantity() {
        return Integer.parseInt(quantityField.getText());
    }

    /**
     * Called when the cart is updated.
     *
     * @param cart the cart that was updated.
     */
    @Override
    public void update(Cart cart) {
        SwingUtilities.invokeLater(() -> {
            updateCartDisplay();
            updateCartIcon();
            updateProductDisplay(Database.getInstance().getProducts()); //
Refresh product display
        });
    }
}


package cop4331.gui;

import javax.swing.*;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

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

/**
 * GUI class representing the login view of the application.
 * Provides fields for entering username and password, along with login and
sign-up buttons.
 */
public class LoginView extends JFrame {
    /** Text field for entering the username. */
    private JTextField usernameField;

    /** Password field for entering the password. */
    private JPasswordField passwordField;

    /** Button for initiating the login process. */
    private JButton loginButton;

    /** Button for navigating to the sign-up view. */
    private JButton signupButton;

    /**
     * Constructs the login view and initializes its components.
     */
    public LoginView() {
        setTitle("Login");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Header
        JLabel header = new JLabel("Welcome to the Shopping Cart App",
JLabel.CENTER);
        header.setFont(new Font("Arial", Font.BOLD, 18));
        add(header, BorderLayout.NORTH);

        // Center Panel
        JPanel centerPanel = new JPanel(new GridLayout(2, 2, 10, 10));
        centerPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
10));
        centerPanel.add(new JLabel("Username:"));
        usernameField = new JTextField();
        centerPanel.add(usernameField);
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        centerPanel.add(new JLabel("Password:"));
        passwordField = new JPasswordField();
        centerPanel.add(passwordField);
        add(centerPanel, BorderLayout.CENTER);

        // Buttons
        JPanel buttonPanel = new JPanel(new FlowLayout());
        loginButton = new JButton("Login");
        signupButton = new JButton("Sign Up");
        buttonPanel.add(loginButton);
        buttonPanel.add(signupButton);
        add(buttonPanel, BorderLayout.SOUTH);

        setLocationRelativeTo(null); // Center the window
    }

    /**
     * Retrieves the entered username.
     *
     * @return the username entered in the text field.
     */
    public String getUsername() {
        return usernameField.getText();
    }

    /**
     * Retrieves the entered password.
     *
     * @return the password entered in the password field.
     */
    public String getPassword() {
        return new String(passwordField.getPassword());
    }

    /**
     * Retrieves the login button.
     *
     * @return the login button.
     */
    public JButton getLoginButton() {
        return loginButton;
    }

    /**
```

```
     * Retrieves the sign-up button.
     *
     * @return the sign-up button.
     */
    public JButton getSignupButton() {
        return signupButton;
    }

    /**
     * Displays an error message in a dialog box.
     *
     * @param message the error message to display.
     */
    public void showError(String message) {
        JOptionPane.showMessageDialog(this, message, "Error",
JOptionPane.ERROR_MESSAGE);
    }
}

package cop4331.gui;

import cop4331.client.Product;
import cop4331.client.Seller;
import cop4331.client.FinancialData;
import cop4331.client.Database;
import cop4331.client.DiscountedProduct;
import cop4331.client.ProductBundle;

import javax.swing.*;
import java.awt.*;
import java.util.List;

/**
 * GUI class representing the seller view in the system.
 * Provides functionality for sellers to view and manage their inventory,
 * add new products, and access financial data.
 */
public class SellerView extends JFrame {
    private Seller seller;
    private JTextArea inventoryDisplay;
    private JTextField productNameField;
    private JTextField productDescriptionField;
    private JTextField productPriceField;
    private JTextField productQuantityField;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
    private JTextField productInvoicePriceField;
    private JButton addProductButton;
    private JButton viewFinancialButton;
    private JButton createBundleButton;
    private JButton applyDiscountButton;

    /**
     * Constructs the seller view with the specified seller.
     *
     * @param seller the seller using this view.
     */
    public SellerView(Seller seller) {
        this.seller = seller;
        setTitle("Seller View - " + seller.getUsername());
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Inventory Display
        inventoryDisplay = new JTextArea();
        inventoryDisplay.setEditable(false);
        JScrollPane inventoryScroll = new JScrollPane(inventoryDisplay);
        inventoryScroll.setBorder(BorderFactory.createTitledBorder("Your
Inventory"));
        add(inventoryScroll, BorderLayout.CENTER);

        // Bottom Panel
        JPanel bottomPanel = new JPanel();
        bottomPanel.setLayout(new BoxLayout(bottomPanel, BoxLayout.Y_AXIS));
// Vertical alignment
        bottomPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
10)); // Padding

        // Input Fields Panel
        JPanel inputPanel = new JPanel(new GridBagLayout());
        inputPanel.setBorder(BorderFactory.createTitledBorder("Add New
Product"));
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5,5,5,5); // Spacing between components
        gbc.anchor = GridBagConstraints.WEST;

        // Row 1: Product Name
        gbc.gridx = 0;
        gbc.gridy = 0;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        inputPanel.add(new JLabel("Product Name:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
        productNameField = new JTextField(20);
        inputPanel.add(productNameField, gbc);

        // Row 2: Description
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.fill = GridBagConstraints.NONE;
        gbc.weightx = 0;
        inputPanel.add(new JLabel("Description:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
        productDescriptionField = new JTextField(20);
        inputPanel.add(productDescriptionField, gbc);

        // Row 3: Price
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.NONE;
        gbc.weightx = 0;
        inputPanel.add(new JLabel("Price:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
        productPriceField = new JTextField(20);
        inputPanel.add(productPriceField, gbc);

        // Row 4: Quantity
        gbc.gridx = 0;
        gbc.gridy = 3;
        gbc.fill = GridBagConstraints.NONE;
        gbc.weightx = 0;
        inputPanel.add(new JLabel("Quantity:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
        productQuantityField = new JTextField(20);
        inputPanel.add(productQuantityField, gbc);

        // Row 5: Invoice Price
        gbc.gridx = 0;
        gbc.gridy = 4;
        gbc.fill = GridBagConstraints.NONE;
        gbc.weightx = 0;
        inputPanel.add(new JLabel("Invoice Price:"), gbc);

        gbc.gridx = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
        productInvoicePriceField = new JTextField(20);
        inputPanel.add(productInvoicePriceField, gbc);

        bottomPanel.add(inputPanel);

        // Buttons Panel
        JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.CENTER,
10, 10));
        addProductButton = new JButton("Add Product");
        viewFinancialButton = new JButton("View Financial Data");
        createBundleButton = new JButton("Create Bundle");
        applyDiscountButton = new JButton("Apply Discount");

        buttonsPanel.add(addProductButton);
        buttonsPanel.add(viewFinancialButton);
        buttonsPanel.add(createBundleButton);
        buttonsPanel.add(applyDiscountButton);

        bottomPanel.add(buttonsPanel);

        add(bottomPanel, BorderLayout.SOUTH);

        setLocationRelativeTo(null);

        // Add action listener to addProductButton
        addProductButton.addActionListener(e -> {
            try {
                String name = getProductName();
                String description = getProductDescription();
                double price = getProductPrice();
                int quantity = getProductQuantity();
```

```
                double invoicePrice = getProductInvoicePrice();

                if (name.isEmpty()) {
                    JOptionPane.showMessageDialog(this, "Product name cannot
be empty!", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if (description.isEmpty()) {
                    JOptionPane.showMessageDialog(this, "Product description
cannot be empty!", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if (price <= 0) {
                    JOptionPane.showMessageDialog(this, "Price must be
greater than 0!", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if (quantity <= 0) {
                    JOptionPane.showMessageDialog(this, "Quantity must be
greater than 0!", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if (invoicePrice <= 0) {
                    JOptionPane.showMessageDialog(this, "Invoice price must
be greater than 0!", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }

                String productId =
String.valueOf(System.currentTimeMillis());
                Product product = new Product(productId, name, description,
price, quantity, seller.getId(), invoicePrice, "product");
                seller.addProduct(product);

                // Save the updated data to products.json
                Database.getInstance().saveData();

                updateInventoryDisplay();

                // Clear input fields
                productNameField.setText("");
                productDescriptionField.setText("");
                productPriceField.setText("");
                productQuantityField.setText("");
```

```
                productInvoicePriceField.setText("");

                JOptionPane.showMessageDialog(this, "Product added
successfully!");

            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this, "Invalid input! Please
ensure price and quantity are numbers.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        });

        // Add action listener to viewFinancialButton
        viewFinancialButton.addActionListener(e -> {
            FinancialData financialData = seller.getFinancialData();
            String message = String.format("Costs: $%.2f\nRevenues:
$%.2f\nProfits: $%.2f",
                    financialData.getCosts(), financialData.getRevenues(),
financialData.getProfits());
            JOptionPane.showMessageDialog(this, message, "Financial Data",
JOptionPane.INFORMATION_MESSAGE);
        });

        // Action listeners for product bundle and discount buttons
        createBundleButton.addActionListener(e -> createBundle());
        applyDiscountButton.addActionListener(e -> applyDiscount());

        // Initial inventory display
        updateInventoryDisplay();
    }

    /**
     * Updates the inventory display to reflect the current products in the
seller's inventory.
     */
    public void updateInventoryDisplay() {
        List<Product> products = seller.getInventory().getProducts();
        inventoryDisplay.setText("");
        for (Product product : products) {
            inventoryDisplay.append("Name: " + product.getName() + "\n");
            inventoryDisplay.append("Description: " +
product.getDescription() + "\n");
            inventoryDisplay.append("Listed Price: $" + product.getPrice() +
"\n");
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
            inventoryDisplay.append("Invoice Price: $" +
product.getInvoicePrice() + "\n");
            inventoryDisplay.append("Stock: " + product.getQuantity() +
"\n");
            inventoryDisplay.append("----------------------------------\n");
        }
    }

    /**
     * Creates a product bundle by selecting multiple products from the
inventory.
     */
    private void createBundle() {
        // Get the list of products from the seller's inventory
        List<Product> products = seller.getInventory().getProducts();

        // Check if there are enough products to create a bundle
        if (products.size() < 2) {
            JOptionPane.showMessageDialog(this, "You need at least two
products to create a bundle.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Create a list of product names for display
        String[] productNames = products.stream()
                .map(product -> product.getId() + ": " + product.getName())
                .toArray(String[]::new);

        // Show a multiple selection dialog to select products for the bundle
        JList<String> productList = new JList<>(productNames);

productList.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        JScrollPane scrollPane = new JScrollPane(productList);
        scrollPane.setPreferredSize(new Dimension(300, 200));

        int result = JOptionPane.showConfirmDialog(this, scrollPane, "Select
Products for Bundle", JOptionPane.OK_CANCEL_OPTION);
        if (result == JOptionPane.OK_OPTION) {
            List<String> selectedValues =
productList.getSelectedValuesList();
            if (selectedValues.size() < 2) {
                JOptionPane.showMessageDialog(this, "Please select at least
two products for the bundle.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
            }

            // Ask for the bundle name
            String bundleName = JOptionPane.showInputDialog(this, "Enter
Bundle Name:");
            if (bundleName == null || bundleName.trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Bundle name cannot be
empty.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Ask for the bundle description
            String bundleDescription = JOptionPane.showInputDialog(this,
"Enter Bundle Description:");
            if (bundleDescription == null ||
bundleDescription.trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Bundle description
cannot be empty.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Create the ProductBundle
            String bundleId = "bundle-" + System.currentTimeMillis();
            ProductBundle bundle = new ProductBundle(bundleId, bundleName,
bundleDescription);

            // Add selected products to the bundle
            for (String value : selectedValues) {
                String productId = value.split(":")[0];
                Product product = products.stream()
                        .filter(p -> p.getId().equals(productId))
                        .findFirst()
                        .orElse(null);
                if (product != null) {
                    bundle.addProduct(product);
                }
            }

            // Add the bundle to the seller's inventory and the database
            seller.addProduct(bundle);
            updateInventoryDisplay();

            JOptionPane.showMessageDialog(this, "Bundle created
successfully!");
```

```
        }
    }

    /**
     * Applies a discount to a selected product.
     */
    private void applyDiscount() {
        // Get the list of products from the seller's inventory
        List<Product> products = seller.getInventory().getProducts();

        if (products.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No products available to
apply discount.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Create a list of product names for display
        String[] productNames = products.stream()
                .map(product -> product.getId() + ": " + product.getName())
                .toArray(String[]::new);

        // Show a selection dialog to select a product
        String selectedProduct = (String) JOptionPane.showInputDialog(this,
"Select a Product:", "Apply Discount",
                JOptionPane.PLAIN_MESSAGE, null, productNames,
productNames[0]);

        if (selectedProduct != null) {
            String productId = selectedProduct.split(":")[0];
            Product product = products.stream()
                    .filter(p -> p.getId().equals(productId))
                    .findFirst()
                    .orElse(null);

            if (product != null) {
                // Ask for the discount rate
                String discountRateStr = JOptionPane.showInputDialog(this,
"Enter Discount Rate (e.g., 0.10 for 10%):");
                try {
                    double discountRate =
Double.parseDouble(discountRateStr);
                    if (discountRate <= 0 || discountRate >= 1) {
                        JOptionPane.showMessageDialog(this, "Discount rate
must be between 0 and 1.", "Error", JOptionPane.ERROR_MESSAGE);
```

```
                        return;
                    }

                    // Create a DiscountedProduct and replace the original
product in the inventory
                    DiscountedProduct discountedProduct = new
DiscountedProduct(product, discountRate);
                    seller.getInventory().removeProduct(product);
                    seller.getInventory().addProduct(discountedProduct);
                    Database.getInstance().saveData();

                    updateInventoryDisplay();

                    JOptionPane.showMessageDialog(this, "Discount applied
successfully!");

                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(this, "Invalid discount
rate.", "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }

    /**
     * Retrieves the product name entered by the seller.
     *
     * @return the entered product name.
     */
    public String getProductName() {
        return productNameField.getText();
    }

    /**
     * Retrieves the product description entered by the seller.
     *
     * @return the entered product description.
     */
    public String getProductDescription() {
        return productDescriptionField.getText();
    }

    /**
     * Retrieves the product price entered by the seller.
```

Grace Obeso-Silva
Thomas Allred
Christian Rodrigues

```
     *
     * @return the entered product price.
     * @throws NumberFormatException if the entered price is not a valid
double.
     */
    public double getProductPrice() {
        return Double.parseDouble(productPriceField.getText());
    }

    /**
     * Retrieves the product quantity entered by the seller.
     *
     * @return the entered product quantity.
     * @throws NumberFormatException if the entered quantity is not a valid
integer.
     */
    public int getProductQuantity() {
        return Integer.parseInt(productQuantityField.getText());
    }

    /**
     * Retrieves the product invoice price entered by the seller.
     *
     * @return the entered product invoice price.
     * @throws NumberFormatException if the entered invoice price is not a
valid double.
     */
    public double getProductInvoicePrice() {
        return Double.parseDouble(productInvoicePriceField.getText());
    }
}
```