E-Commerce System-Documentation

Name: Khong Dinh Tu

ID: 24110145

---

1. Object-Oriented Analysis (OOA)

Following the 4-step OOA model, the system has the following objects:

1.1 Objects

- Product
- ElectricProduct (inherits from Product)
- ShoppingCart
- Order
- Invoice
- Shopee (manages products and orders)

1.2 Attributes for Each Object

- **Product:** id, name, category, weight, price, stock
- **ElectricProduct:** warrantyMonths (inherits attributes from Product)
- **ShoppingCart:** itemList (InventoryList<Product*>), total
- **Order:** orderID, items, total
- **Invoice:** items, shippingFee
- **Shopee:** inventory, orders, nextOrderId

1.3 Methods

- **Product:** displayInfo(), applyDiscount(), updateStock(), reduceStock(), getId(), getName(), getPrice(), operator==, operator<
- **ElectricProduct:** displayInfo() (override), updateStock() (override), applyDiscount() (override)
- **ShoppingCart:** operator+=, operator-=, displayCart(), applyDiscount(), checkout(), getTotal()
- **Order:** display()
- **Invoice:** calcTotal(), displayInvoice()
- **Shopee:** addProduct(), showInventory(), findProduct(), createOrder(), showOrders()

1.4 Inheritance Relationships

- **Product** is the base class.
- **ElectricProduct** inherits from Product.
- **Discountable** is an abstract class (interface) implemented by both Product and ShoppingCart

2. Class Design

2.1 Encapsulation

- All attributes such as id, price, and stock in Product are private/protected.
- Access is only provided via getters (e.g., getPrice()) and controlled methods like updateStock() and reduceStock().

2.2 Inheritance

- ElectricProduct extends Product, adding a warrantyMonths attribute.
- Methods like displayInfo(), updateStock(), and applyDiscount() are overridden.

2.3 Polymorphism

- displayInfo() is declared as virtual in Product and overridden in ElectricProduct.
- applyDiscount() is pure virtual in Discountable and is implemented differently in Product and ShoppingCart.

2.4 Operator Overloading

- operator== → compares two products based on weight.
- operator< → compares two products based on price.
- operator+= → adds a product to the shopping cart.
- operator-= → removes a product from the shopping cart.

3. Code Walkthrough

Operator Overloading

- cart += product; → adds a product to the cart, reduces its stock, and increases the cart total.
- cart -= product; → removes a product from the cart and adjusts total.
- if (*book == *phone) → compares two products' weight.
- if (*book < *laptop) → compares product prices.

Example Flow (ShoppingCart → Order → Invoice)

- A user adds products to the cart using +=.

- The checkout() method is called, returning a list of products and the total price.
- Shopee creates an Order object from the checkout result.
- An Invoice object is generated to display order details with a shipping fee.

4. Sample Output

When running the code, the ouput includes:

```
=== Test 1: Show Inventory ===
=== Inventory (3 items) ===
[ID]: 101
[Name]: C++ Primer
[Category]: Book
[Weight]: 500 g
[Price]: 150000 VND
[Stock]: 5
[ID]: 102
[Name]: Gaming Laptop
[Category]: Electronics
[Weight]: 2500 g
[Price]: 20000000 VND
[Stock]: 3
[Warranty Months]: 24 months
[ID]: 103
[Name]: Smartphone
[Category]: Electronics
[Weight]: 180 g
[Price]: 12000000 VND
[Stock]: 0
```

5. UML Diagrams

5.1 Class Diagram

- **Product** (abstract methods from Discountable)

- **ElectricProduct** inherits Product
- **ShoppingCart** implements Discountable
- **Order** and **Invoice** aggregate Products
- **Shopee** aggregates InventoryList<Product*> and Orders

Relationships:

- Inheritance: Product → ElectricProduct
- Interface realization: Discountable → Product, ShoppingCart
- Aggregation: Shopee → Orders, InventoryList
- Association: ShoppingCart → Product

5.2 Sequence Diagram (Checkout → Order)

Actors: User → ShoppingCart → Shopee → Order → Invoice

Steps:

1. User adds products to the cart with +=.
2. User calls checkout() on ShoppingCart.
3. ShoppingCart sends checkout result to Shopee.
4. Shopee creates an Order with items and total.
5. Shopee generates an Invoice with shipping fee.
6. Invoice displays order details back to the User.

6. Use of LLM (ChatGPT)

I used ChatGPT for:

- Brainstorming the design of the template class InventoryList.
- Suggesting intuitive operator overloads (`+=`, `-=`).

- Refining test case scenarios (out-of-stock products, restocking, multiple orders).
- Guidance on UML notation for class and sequence diagrams.

Example Prompt:
"Can you suggest test cases for an OOP-based C++ E-commerce cart and order system with operator overloading?"

Response:
ChatGPT suggested:

- *Attempting to add out-of-stock products to the cart.*
- *Adding the same product multiple times and displaying quantities correctly.*
- *Checking out twice and verifying correct stock updates.*
- *Restocking products and testing overridden methods in ElectricProduct.*

All final code and documentation were written and verified personally.