Documentation Report
E-Commerce Product Management System

Tran Le Dung
MSSV: 24110084

September 21, 2025

1 Object-Oriented Analysis (OOA)

The goal of the system is to simulate a simplified e-commerce platform that manages products, shopping carts, and orders while demonstrating object-oriented programming (OOP) concepts.

Actors

- **Customer:** interacts with the system by selecting products, adding them to a cart, and placing orders.
- **System:** manages product inventory, stock updates, discounts, and order processing.

Use Cases

- Add product to inventory.
- Add/remove product from cart.
- Apply discounts to products or the whole cart.
- Process and cancel orders.
- Display product, cart, and order details.

Key Responsibilities

- **Product:** Encapsulates attributes (id, name, price, stock, description).
- **Electronics:** Specialization of product with brand, model, warranty.
- **ShoppingCart:** Maintains a list of products, calculates totals, applies discounts.
- **Order:** Records finalized purchases with date, status, and items.
- **InventoryList¡T¿:** Generic storage for products or categories.

2 Class Design Explanation

Inheritance

- `Electronics` inherits from `Product`. This allows reuse of product features and extension with warranty/brand/model. Example: `updateStock()` is overridden in `Electronics` to add special logging.

Interfaces

- The `Discountable` interface declares `applyDiscount()`.
- `Product` and `ShoppingCart` both implement this, but in different ways:
  - Products apply discount individually to their price.
  - Shopping carts apply discount to the total sum of items.

  This demonstrates polymorphism with a common contract.

Operator Overloading

- `Product::operator==` and `!=` compare products by their ID. This makes product equality intuitive.
- `ShoppingCart::operator+=` allows adding products to a cart in natural syntax: `cart += laptop;`

Template Class

- `InventoryList<T>` is a generic container built on top of `std::vector`.
- Used for:
  - `InventoryList<Product*>`: manages stock and cart contents.
  - `InventoryList<string>`: manages product categories.
- Provides reusable methods: add, remove, search, display, clear, and `operator[]`.

3 Code Walkthrough

Product Class

- Encapsulates attributes with validation (negative price/stock corrected to 0).
- Implements `applyDiscount()` from `Discountable`.
- Provides equality operators to compare products by ID.

Electronics Class

- Extends `Product` with brand, model, warranty.
- Overrides `updateStock()` to log additional behavior.
- Adds custom method `extendWarranty()`.

ShoppingCart Class

- Maintains an `InventoryList<Product*>` of items.
- Overloads `+=` operator to add items.
- Implements `applyDiscount()` differently from products (applies to total).
- Includes operations: remove, clear, calculate total, display cart.

Order Class

- Created from a shopping cart snapshot.
- Tracks order ID, date, status, and items.
- Provides `processOrder()` and `cancelOrder()`.

InventoryList

- Template-based reusable collection.
- Demonstrates type generalization in C++.
- Used with both complex (pointers to products) and simple (strings) types.

4 Test Results

The `main()` function validates all system features:

1. **Product Creation:** Products and electronics instantiated with input validation.
2. **Template Test:** InventoryList handles multiple product types.
3. **Inheritance:** Virtual methods confirm polymorphism.
4. **Operator Overloading:** Products compared with `==`, cart items added with `+=`.
5. **Interface:** Discounts applied on both product and cart.
6. **Order Management:** Orders created from cart, processed, and displayed.
7. **Error Handling:** Out-of-stock and invalid discount rates tested.
8. **Extra Features:** Cart removal/clearing and warranty extension tested.

Listing 1: Sample Output Snippet

```
1. CREATING PRODUCTS:
Products created successfully!

2. TESTING TEMPLATE CLASS:
Inventory size: 4
Index 0: Product ID: 1, Name: Gaming Laptop, Price: $1500.00 ...

4. TESTING OPERATOR OVERLOADING:
Comparing products using == operator:
Laptop and mouse are different products

Testing += operator with ShoppingCart:
Product 'Gaming Laptop' added to cart successfully!
Cart total: $1500.00
...
Cart Discount Applied:
Original total: $1670.00
Discount (10%): -$167.00
Final total: $1503.00
```

These outputs confirm correct OOP behavior and system functionality.

5 LLM Usage

I used ChatGPT as a supportive tool during the project:

- To brainstorm ideas for the `InventoryList<T>` template.
- To refine operator overloading design for `ShoppingCart`.
- To draft documentation structure and LaTeX formatting.

Example prompt: *"Suggest a template class for inventory in C++."*

Response summarized the use of `std::vector` with add/remove/search operations, which I adapted to the final implementation.