I-OOA Model – E-commerce Product Management System

1. Identified Objects

Trong hệ thống quản lý sản phẩm trực tuyến, các đối tượng chính được nhận diện bao gồm:

- **Product**: đại diện cho một sản phẩm chung trong cửa hàng.
- **Electronics**: sản phẩm điện tử, kế thừa từ Product, có thêm thuộc tính riêng như thời gian bảo hành.
- **Clothing**: sản phẩm quần áo, kế thừa từ Product, có thuộc tính kích cỡ.
- **ShoppingCart**: giỏ hàng chứa các sản phẩm mà khách hàng muốn mua.
- **Order**: đơn hàng được tạo từ giỏ hàng.
- **InventoryList**: danh sách sản phẩm, sử dụng template để có thể lưu trữ nhiều loại đối tượng khác nhau.
- **Discountable**: interface trừu tượng để định nghĩa hành vi áp dụng giảm giá.

2. Attributes of Objects

- **Product** có các thuộc tính: id, name, price, và stock để lưu thông tin cơ bản về sản phẩm.
- **Electronics** bổ sung thuộc tính warrantyMonths để lưu số tháng bảo hành.
- **Clothing** bổ sung thuộc tính size để xác định kích cỡ quần áo.
- **ShoppingCart** lưu trữ danh sách các CartItem (mỗi CartItem gồm một sản phẩm và số lượng) và tổng giá trị total.
- **Order** có orderId và chứa một ShoppingCart.
- **InventoryList** sử dụng vector template items để quản lý danh sách các sản phẩm.

3. Methods of Objects

- **Product**: có các phương thức display() để hiển thị thông tin, updateStock(int) để cập nhật tồn kho, applyDiscount(double) để áp dụng giảm giá, và operator== để so sánh sản phẩm theo ID.
- **Electronics**: override phương thức display() và updateStock(int) để hiển thị thông tin đặc thù và xử lý cập nhật tồn kho cho sản phẩm điện tử.
- **Clothing**: override display() để hiển thị thông tin với size.
- **ShoppingCart**: hỗ trợ operator+= để thêm sản phẩm vào giỏ, display() để hiển thị giỏ hàng, và applyDiscount(double) để giảm giá cho tổng tiền.
- **Order**: phương thức display() hiển thị chi tiết đơn hàng.
- **InventoryList**: các phương thức add(), remove(), get(), size() và displayAll() để quản lý danh sách sản phẩm.
- **Discountable**: pure virtual function applyDiscount(double) được các lớp Product và ShoppingCart cài đặt.

4. Inheritance Relationships

- **Electronics** và **Clothing** đều kế thừa **Product** (single inheritance).
- **Product** và **ShoppingCart** thực hiện interface **Discountable**, cho phép polymorphic áp dụng giảm giá.
- Các phương thức như display() và updateStock() được override để thể hiện tính đa hình.

5. OOP Concepts Applied

- **Encapsulation**: tất cả thuộc tính là private, truy cập qua getter/setter.
- **Inheritance**: mở rộng Product thành Electronics và Clothing.
- **Polymorphism / Interface**: Discountable cho phép applyDiscount thực thi đa hình.
- **Operator Overloading**: operator== trong Product, operator+= trong ShoppingCart.
- **Template Class**: InventoryList<T> quản lý danh sách sản phẩm một cách generic, có thể dùng cho Product*, Electronics*, v.v.
- **Realistic Modeling**: Hệ thống mô phỏng các hoạt động thực tế của cửa hàng trực tuyến như quản lý sản phẩm, giỏ hàng, đơn hàng và áp dụng giảm giá.

II- Class Design Description

1. Product (Base Class)

Lớp Product đại diện cho sản phẩm cơ bản trong cửa hàng. Nó chứa các thuộc tính id, name, price và stock. Product thực hiện interface Discountable, vì vậy có phương thức applyDiscount(double rate) để tính giá sau khi giảm. Ngoài ra, lớp này còn có các phương thức:

- display(): hiển thị thông tin sản phẩm.
- updateStock(int quantity): cập nhật tồn kho (cộng hoặc trừ số lượng).
- operator==: so sánh hai sản phẩm theo ID.

Product được thiết kế với các phương thức ảo (virtual) để cho phép lớp con override và thể hiện tính đa hình.

2. Electronics (Derived Class)

Electronics kế thừa từ Product và bổ sung thuộc tính riêng warrantyMonths.
Các phương thức được override:

- display(): hiển thị thông tin sản phẩm kèm bảo hành.
- updateStock(int quantity): override để thêm thông báo đặc thù khi cập nhật tồn kho điện tử.

Lớp này minh họa **single inheritance** và **method overriding.**

3. Clothing (Derived Class)

Clothing cũng kế thừa từ Product và thêm thuộc tính size (kích cỡ quần áo).
Override phương thức:

- display(): hiển thị thông tin sản phẩm kèm kích cỡ.

Lớp này đơn giản nhưng minh họa cách mở rộng Product cho các loại sản phẩm khác nhau.

4. Discountable (Interface / Abstract Class)

Discountable là interface trừu tượng định nghĩa hành vi giảm giá:

- applyDiscount(double rate) là **pure virtual function.**
  Các lớp Product và ShoppingCart implement interface này, cho phép giảm giá polymorphic cho từng sản phẩm riêng lẻ hoặc toàn bộ giỏ hàng.

5. ShoppingCart

Lớp ShoppingCart lưu danh sách các CartItem (mỗi CartItem gồm một shared_ptr<Product> và số lượng). Các tính năng chính:

- operator+=: thêm sản phẩm vào giỏ, tự động tăng số lượng nếu sản phẩm đã có.
- display(): hiển thị toàn bộ giỏ hàng và tổng tiền.
- applyDiscount(double rate): áp dụng giảm giá cho tổng giá trị giỏ hàng.

ShoppingCart thực hiện interface Discountable, cho phép áp dụng giảm giá polymorphic.

6. Order

Lớp Order chứa một ShoppingCart và orderId. Phương thức display() hiển thị chi tiết đơn hàng.

Lớp này mô phỏng quy trình thực tế khi khách hàng hoàn tất mua hàng.

7. InventoryList (Template Class)

InventoryList<T> là template class quản lý danh sách sản phẩm (hoặc bất kỳ loại đối tượng nào). Các phương thức:

- add(T item): thêm đối tượng.
- remove(int index): xóa đối tượng theo chỉ số.
- get(int index): truy xuất đối tượng.
- displayAll(): hiển thị tất cả đối tượng.

Template này cho phép quản lý **generic collection**, dễ dàng mở rộng cho nhiều loại sản phẩm khác nhau.

III- Code Walkthrough / Key Features

1. **Creating Products**:
   Trong main(), các đối tượng Product, Electronics, Clothing được tạo với thông tin cụ thể, minh họa **object instantiation** và **constructor usage**.
2. **Inventory (Template Usage)**:
   InventoryList<shared_ptr<Product>> inventory; sử dụng template class để lưu trữ danh sách sản phẩm, sau đó gọi inventory.displayAll() để hiển thị toàn bộ inventory.
3. **ShoppingCart Operations (Operator Overloading)**:
   - cart += p1; sử dụng **operator+=** để thêm sản phẩm vào giỏ.
   - Khi thêm sản phẩm cùng ID, số lượng tự động tăng.
   - Nếu sản phẩm **hết stock**, chương trình báo lỗi nhưng không thêm vào giỏ hàng.
4. **Discount Application (Interface / Polymorphism)**:
   - cart.applyDiscount(0.1); áp dụng giảm giá 10% cho toàn bộ giỏ hàng.
   - Product::applyDiscount() có thể áp dụng giảm giá riêng lẻ cho từng sản phẩm.
5. **Operator== Usage**:
   - So sánh hai sản phẩm: (*p1 == *p2) để xác định chúng có cùng ID hay không.
6. **Order Handling**:
   - Order order1("O001", cart); tạo đơn hàng từ giỏ hàng hiện tại.
   - order1.display(); hiển thị chi tiết đơn hàng, tổng tiền, danh sách sản phẩm.
7. **Polymorphism and Method Overriding**:

- Gọi display() hoặc updateStock() trên con trỏ Product* sẽ thực thi phiên bản override trong Electronics hoặc Clothing nếu đối tượng là lớp con.

8. **Encapsulation**:
   - Thuộc tính private, chỉ truy cập thông qua getter/setter hoặc phương thức công khai.
   - Tổng tiền, số lượng trong giỏ hàng không thể thay đổi trực tiếp từ bên ngoài.

IV-Testing and Output Explanation

Để kiểm tra hệ thống, chương trình đã tạo một số sản phẩm mẫu (Book, Laptop, T-Shirt), đưa chúng vào inventory, thêm vào giỏ hàng, áp dụng giảm giá, so sánh sản phẩm, và cuối cùng tạo một đơn hàng. Các test case được thiết kế để chứng minh từng yêu cầu quan trọng trong đề bài.

1. Inventory Display (Template Class Test)

Output:

```
--- INVENTORY ---
Product [P01] Book - $10 | Stock: 5
Electronics [E01] Laptop - $1200 | Stock: 2 | Warranty: 24 months
Clothing [C01] T-Shirt - $20 | Stock: 3 | Size: L
```

- InventoryList<shared_ptr<Product>> được sử dụng để lưu trữ nhiều loại sản phẩm khác nhau.
- Phương thức displayAll() gọi hàm display() polymorphic, cho thấy tính đa hình (Laptop hiển thị kèm bảo hành, T-Shirt kèm kích cỡ).

Chứng minh template class và polymorphism hoạt động đúng.

2. Adding Products to Cart (Operator Overloading and Stock Update)

Output:

```
(Electronics stock update includes fragile handling)
(Electronics stock update includes fragile handling)
Cannot add Laptop (out of stock)
```

- cart += p1; cart += p2; cart += p3; cart += p2; cart += p2; kiểm tra **operator+=**.
- Laptop được thêm 2 lần, sau đó lần thứ 3 không thành công vì hết tồn kho → in ra thông báo lỗi.
- Khi thêm Electronics, updateStock() được override và in thêm thông báo xử lý đặc biệt.

Chứng minh operator overloading và method overriding.

3. Cart Display Before Discount

Output:

```
--- CART BEFORE DISCOUNT ---
=== Cart Contents ===
1x Product [P01] Book - $10 | Stock: 4
2x Electronics [E01] Laptop - $1200 | Stock: 0 | Warranty: 24 months
1x Clothing [C01] T-Shirt - $20 | Stock: 2 | Size: L
Total: $2430
```

- Giỏ hàng hiển thị chính xác số lượng, tồn kho sau khi cập nhật, và tổng tiền ban đầu.

Chứng minh giỏ hàng tính toán đúng tổng tiền và quản lý số lượng.

4. Discount Application (Interface Test)

Output:

```
Applying 10% discount...
=== Cart Contents ===
1x Product [P01] Book - $10 | Stock: 4
2x Electronics [E01] Laptop - $1200 | Stock: 0 | Warranty: 24 months
1x Clothing [C01] T-Shirt - $20 | Stock: 2 | Size: L
Total: $2187
```

- Gọi cart.applyDiscount(0.1); giảm tổng tiền 10%.
- Từ $2430 còn $2187, chứng minh interface Discountable được implement đúng.

Chứng minh interface và polymorphic discounting.

5. Product Comparison (Operator== Test)

Output:

```
Compare p1 and p2: different
```

- (*p1 == *p2) trả về false, vì Book và Laptop có ID khác nhau.
- Nếu so sánh 2 sản phẩm cùng ID, kết quả sẽ là “same”.

Chứng minh operator== hoạt động đúng.

6. Order Creation and Display

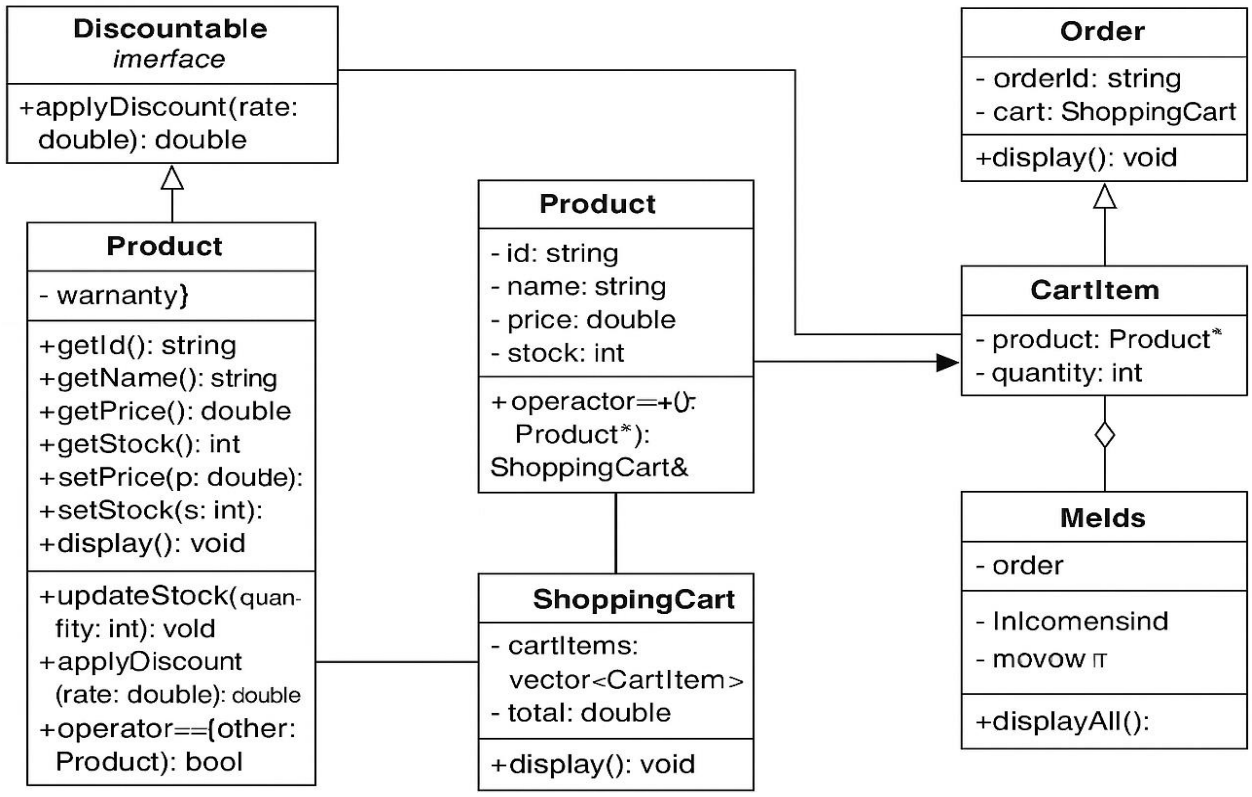
Output:

```
=== Order O001 ===
=== Cart Contents ===
1x Product [P01] Book - $10 | Stock: 4
2x Electronics [E01] Laptop - $1200 | Stock: 0 | Warranty: 24 months
1x Clothing [C01] T-Shirt - $20 | Stock: 2 | Size: L
Total: $2187
```

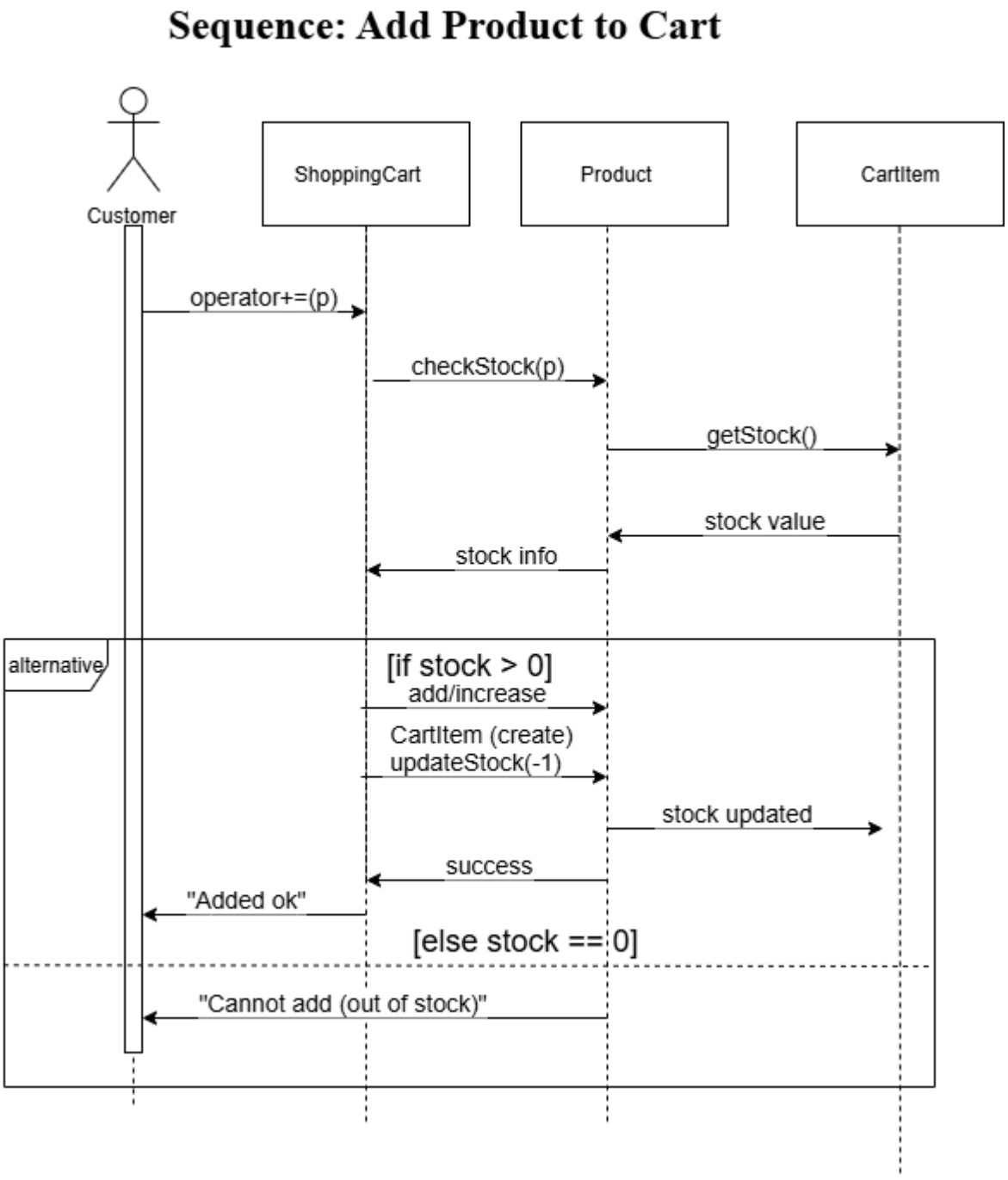
- Order chứa giỏ hàng và in ra chi tiết đơn hàng với mã O001.
- Thông tin hiển thị trùng khớp với giỏ hàng trước đó, chứng minh dữ liệu được lưu và chuyển đúng.

Chứng minh lớp Order và quy trình mua hàng cơ bản hoạt động.

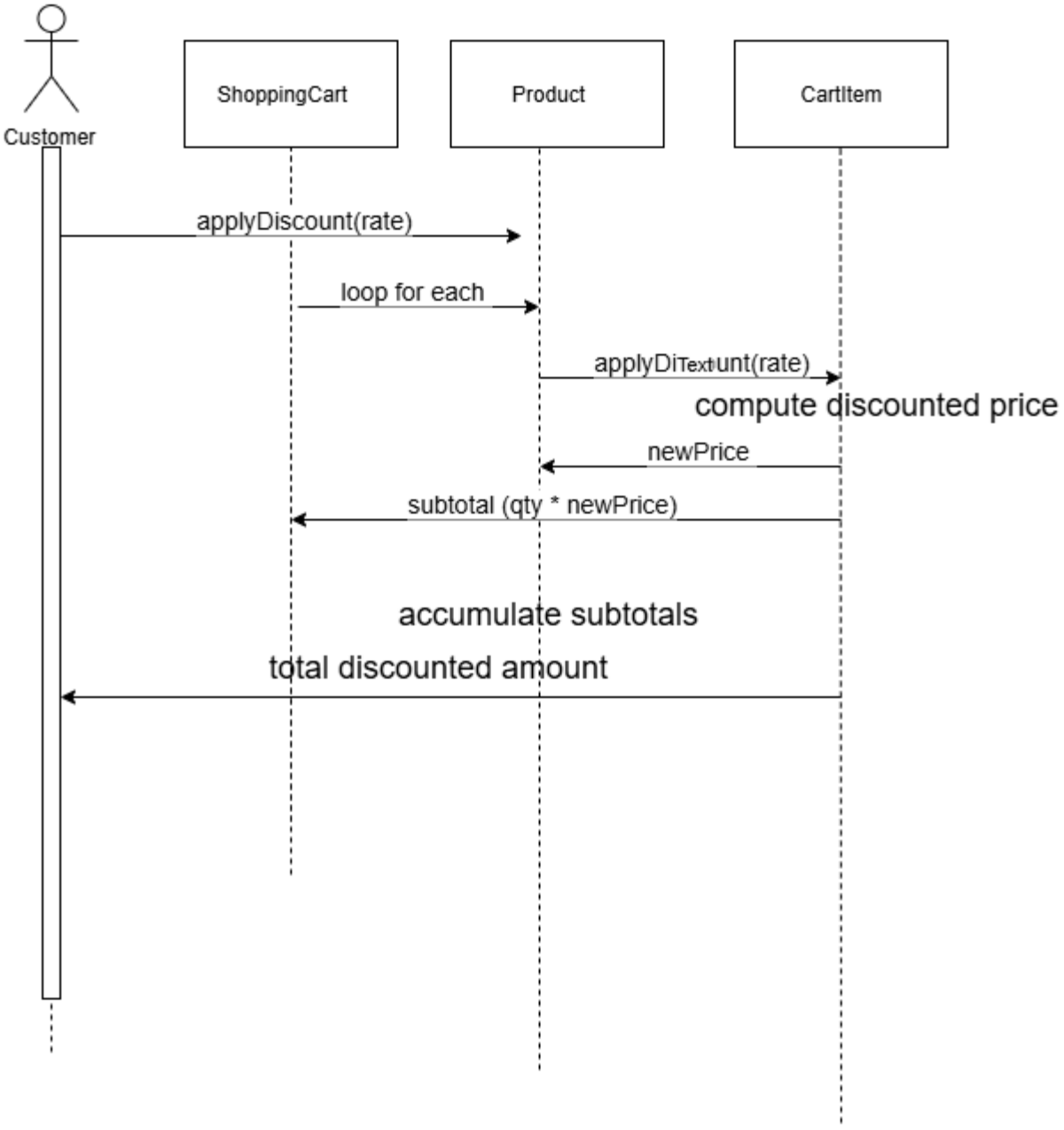
V-UML CLASS DIAGRAM



VI-UML SEQUENCE DIAGRAM



Sequence: Apply Discount (cart-wide)



VII- Use of LLM AI Model

To support the development process, I incorporated an LLM (Large Language Model), specifically ChatGPT, in an **ethical and limited manner** as follows:

- **Brainstorming ideas:**
  - Discussed possible class structures during the Object-Oriented Analysis (OOA) phase.
  - Explored ideas for applying OOP principles (inheritance, polymorphism, operator overloading, templates) to the system.
- **Debugging assistance:**
  - Asked for suggestions when encountering compiler errors or logical issues (e.g., const-correctness problems).
  - Received guidance on how to fix type-related errors without changing the intended logic.
- **Concept explanation:**
  - Used it to clarify how operator overloading works and how == can be used to compare product objects by ID.
  - Reviewed how to design a Cart class to group identical products using operator overloading.

Important Note:

I **did not use the LLM to generate the entire code, UML diagrams, or full documents.**

All source code and design decisions were implemented by me. The LLM was only used as a **supporting tool for understanding and brainstorming**, in compliance with ethical usage guidelines.