Ergi Demiraj

Project Report

Description/Overview

A shopping cart system where users can add, remove, and view products. There are two types of products: Digital and Physical. Users can also apply two types of discounts: Percentage-based or Fixed Amount.

I used Object-Oriented Programming principles like Inheritance, Encapsulation, Polymorphism, and Abstraction. The main classes are **Product, Cart, User, and Discount**. Digital and Physical products extend the Product class. The Cart holds items and calculates the total. The Discount class applies different discounts.

This project helped me understand how to structure a system using OOP concepts.

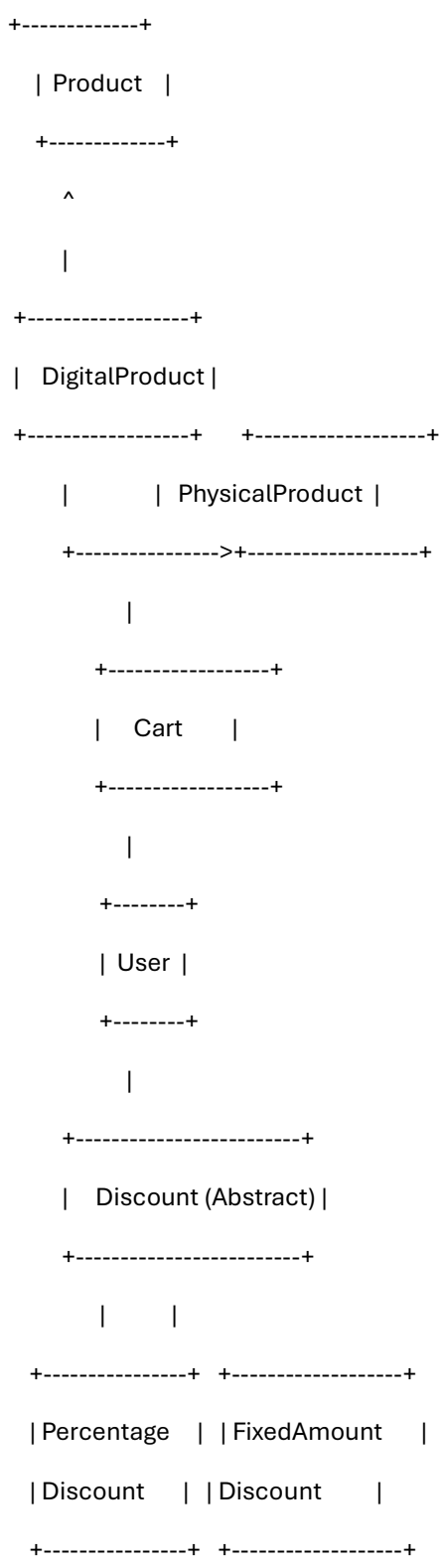
Instructions

The program has four main classes: Product, Cart, User, and Discount. Each class has a specific role in managing the shopping cart system.

- Product Class: Represents both Digital and Physical products, which inherit from it. Digital products have additional attributes like file size and a download link, while Physical products have attributes like weight and shipping cost.
- Cart Class: Stores products for each user, allowing them to add, remove, and view items. It also calculates the total price and applies discounts.
- User Class: Each user has a Cart. Users can add and remove items from their cart and proceed to checkout.
- Discount Class: An abstract class with two types of discounts: PercentageDiscount and FixedAmountDiscount.

Ergi Demiraj

Structure



1. The product class has the attributes of the products, can update the quantity and retrieve the product information.
2. The DigitalProduct class is inherited from the Product class so that it can include digital products as well, and it has some extra attributes.
3. The PhysicalProduct class is inherited from the Product class so that it can include physical products and also has some extra attributes that differ from the digital product.
4. The cart class creates a private list of cart items to store the products, and they are different from each user.
5. The User Class has the user which will have the cart.
6. The Discount class is an abstract class which has the method for the discount.
7. & 8. Then either the Percentage discount or the fixed discount will be chosen which will apply the discount based on a percentage of the total amount or a fixed amount.

Verification of Sanity of Code

To ensure the code works correctly, I tested it with the following sample scenario:

1. Created:
    - 2 DigitalProduct instances.
    - 3 PhysicalProduct instances.
2. Created 2 User instances:
    - User 1 added digital products to the cart.
    - User 2 added physical products to the cart.
3. Verified each user's cart using view_cart().
4. Applied discounts:
    - User 1: PercentageDiscount.
    - User 2: FixedAmountDiscount.
5. Performed checkout:
    - Ensured the total was correct after applying discounts.
    - Confirmed cart was emptied after checkout.

Ergi Demiraj

```
# Creating Digital Products
digital1 = DigitalProduct(101, "E-Book", 10.0, 1, 5, "ebook.com/download")
digital2 = DigitalProduct(102, "Music Album", 15.0, 1, 50, "music.com/download")

# Creating Physical Products
physical1 = PhysicalProduct(201, "Laptop", 1000.0, 1, 5, "15x10x1", 20.0)
physical2 = PhysicalProduct(202, "Headphones", 200.0, 1, 1, "5x5x3", 5.0)
physical3 = PhysicalProduct(203, "Keyboard", 50.0, 1, 2, "18x6x1", 10.0)


# Creating Users
user1 = User(1, "Alice")
user2 = User(2, "Bob")

# Adding Digital Products to User 1
user1.add_to_cart(digital1)
user1.add_to_cart(digital2)

# Adding Physical Products to User 2
user2.add_to_cart(physical1)
user2.add_to_cart(physical2)
user2.add_to_cart(physical3)
```

```
User 1 Cart:
ID: 101, Name:E-Book, Price: $10.0,Quantity: 1, File Size:5MB, Download Link:ebook.com/downloadID: 102, Name:Music Album, Price: $15.0,Quantity: 1, File Size:50MB, Download Link:music.com/download

User 2 Cart:
ID: 201, Name:Laptop, Price: $1000.0,Quantity: 1, Weight: 5lb, Dimensions: 15x10x1, Shipping Cost: $20.0ID: 202, Name:Headphones, Price: $200.0,Quantity: 1, Weight: 1lb, Dimensions: 5x5x3, Shipping Cost: $5.0ID: 203, Name:Keyboard
, Price: $50.0,Quantity: 1, Weight: 2lb, Dimensions: 18x6x1, Shipping Cost: $10.0
```

```
percentage_discount = PercentageDiscount(10)  # 10% Discount
fixed_discount = FixedAmountDiscount(50)  # $50 Discount

# Applying Discounts
user1_total_after_discount = user1.cart.apply_discount(percentage_discount)
user2_total_after_discount = user2.cart.apply_discount(fixed_discount)

print(f"\nUser 1 Total After 10% Discount: ${user1_total_after_discount}")
print(f"User 2 Total After $50 Discount: ${user2_total_after_discount}")

print("\nUser 1 Checkout:")
print(user1.checkout())

print("\nUser 2 Checkout:")
print(user2.checkout())

# Verifying carts are empty after checkout
print("\nUser 1 Cart After Checkout:")
print(user1.cart.view_cart())

print("\nUser 2 Cart After Checkout:")
print(user2.cart.view_cart())
```

```
User 1 Total After 10% Discount: $22.5
User 2 Total After $50 Discount: $1200.0

User 1 Checkout:
Total amount: $25.0. Cart is now empty

User 2 Checkout:
Total amount: $1250.0. Cart is now empty
```

```
User 1 Cart After Checkout:
Cart is empty

User 2 Cart After Checkout:
Cart is empty
```

Ergi Demiraj

Conclusion

Findings:

Product Class: The Product class was simple, with just attributes for ID, name, price, and quantity. It helped set up the base for other product types.

Super and Inheritance: I learned that super().__init__ is important for child classes to use the attributes and methods from the parent class, like in DigitalProduct and

PhysicalProduct.

Removing Products from Cart: I used break in the remove_product method to avoid removing more than one product at once, preventing errors.

Discounts and Polymorphism: I was confused at first about how to apply different discounts, but using polymorphism helped. Now, the cart can handle both percentage and fixed amount discounts.

Abstract Discount Class: Using an abstract class for discounts was helpful to make sure each discount type followed the same structure.

Challenges:

Polymorphism: Understanding how to apply polymorphism to handle multiple discount types was tricky but important for flexibility.

Abstract Classes: I had to learn how to use abstract classes correctly, especially knowing that you can’t instantiate them directly.

Cart Operations: It was a challenge to make sure products were added or removed correctly from the cart.

Limitations and Improvements:

Error Handling: The code doesn't handle errors well, like when trying to remove a product that isn't in the cart.

Discount Checks: There should be checks to ensure discounts aren’t more than the total price.