Gefen Efrati

CMPSC-132 Project 1 Report

Description

This project's purpose is to design and implement a digital inventory management system for representing different types of products and discount interaction with those products. The system utilizes object-oriented programming principles, including encapsulation, inheritance, and polymorphism, to ensure modular and efficient code design.

main.py contains multiple classes that represent different aspects of an inventory system. A base class, *Product* Handles general product attributes like name, price, and quantity; it possesses 2 subclasses for representing digital and physical products. The *Cart* class manages adding, removing, and calculating the total cost of products. The *User* represents a user with an associated shopping cart. *Discount* is an abstract base class that provides an interface for applying discounts; it has 2 subclasses for percentage and fixed amount discounts.

Instructions

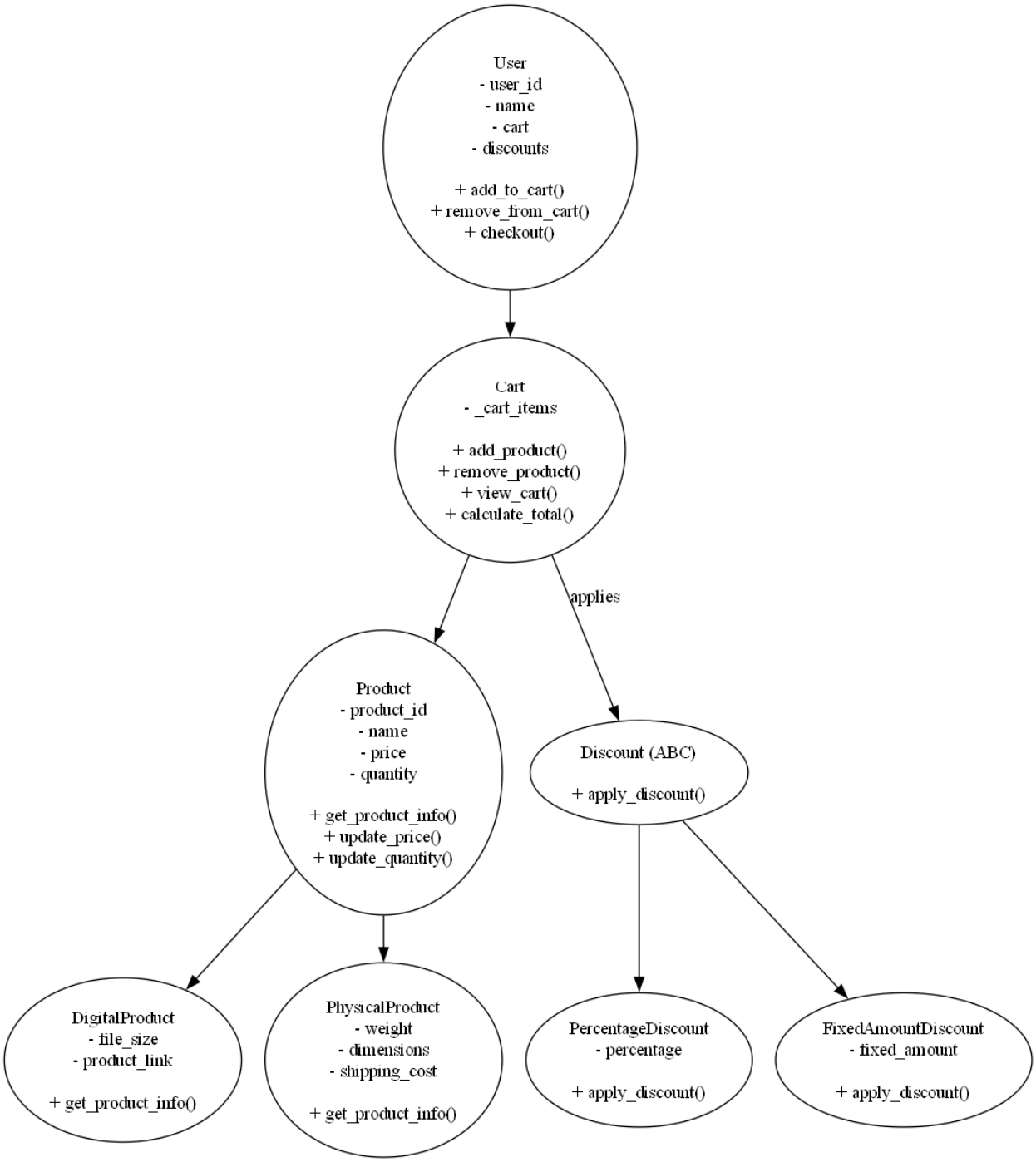
Using the system is straightforward. Products can be created by providing an ID, name, price, and optional attributes, depending on whether they are digital or physical. Once created, the details of a product can be accessed through `get_product_info()`, and modifications can be made using `update_price(new_price)` or `update_quantity(new_quantity)`.

For shopping cart management, a Cart object must be created. Products can be added with `add_product(product)` and removed using `remove_product(product_id)`. The cart's contents can always be checked with `view_cart()`, and the total cost—including any applicable discounts—can be calculated using `calculate_total(discounts=None)`.

Each user in the system has their own cart, managed by the *User* class. Users can add or remove items from their cart and complete their purchase using `checkout()`, which applies discounts and then clears the cart.

Discounts are flexible and can be added to a user's purchase before checkout. A *PercentageDiscount* reduces the total by a set percentage, while a *FixedAmountDiscount* subtracts a fixed amount from the total price.

Structure



Brief class summary

Product Class: This is the base class that represents a general product in the inventory. It includes attributes such as product ID, name, price, and quantity. It also provides methods for updating product details and retrieving product information.

DigitalProduct Class: This class inherits from *Product* and represents digital items. It introduces additional attributes such as file size and a product link, which are essential for downloadable items.

PhysicalProduct Class: Extending from *Product*, this class is designed for physical goods. It includes attributes like weight, dimensions, and shipping costs, making it suitable for tangible items that require shipping.

Cart Class: The cart class serves as the backbone of the shopping experience. It allows users to add and remove products, view their cart contents, and calculate the total cost of their items, including shipping for physical products.

User Class: The user class connects a person to their shopping cart. Each user has their own cart instance, enabling them to manage their purchases individually. Users can add and remove products and proceed to checkout when ready.

Discount Class: This is an abstract base class that defines the structure for discount mechanisms. It enforces a standard interface for applying discounts.

PercentageDiscount Class: A subclass of *Discount*, this class applies a percentage-based reduction to the total cart price.

FixedAmountDiscount Class: Another subclass of *Discount*, this class subtracts a fixed amount from the total price, ensuring that discounts are applied in a straightforward manner.

Verification of Code Sanity

The test cases included in *main.py* thoroughly verify the functionality of the system.

Testing the Product Class

The `test_product_class()` function confirms that product instances are created correctly and that their attributes match expectations. Additionally, it verifies that methods for updating product quantity and retrieving product information function as intended.

```
Product creation works correctly.
update_quantity works correctly.
get_product_info works correctly.
```

Testing Digital and Physical Product Classes

The `test_digital_and_physical_product_classes()` function ensures that both DigitalProduct and PhysicalProduct classes inherit as intended from Product. It checks that the extra attributes specific to each subclass (such as file size for digital products

and shipping cost for physical products) are correctly initialized and accessible. The overridden `get_product_info()` methods are also validated.

```
DigitalProduct creation works correctly.
DigitalProduct.get_product_info works correctly.
PhysicalProduct creation works correctly.
PhysicalProduct.get_product_info works correctly.
```

Testing the Cart Class

The `test_cart_class()` function verifies that items can be added to and removed from the cart. It checks whether the cart correctly maintains its list of items and whether `view_cart()` accurately displays the contents. The `calculate_total()` function is tested to ensure that the total price includes shipping costs for physical products.

```
add_product works correctly.
remove_product works correctly.
view_cart works correctly.
calculate_total works correctly.
cart_items is private and only accessible through public methods.
```

Testing the User Class and Checkout Process

The `test_cart_operations()` function simulates a real shopping scenario. It confirms that users can add and remove items from their carts, view their cart contents, and successfully complete a checkout. It also validates that the cart is emptied after checkout and that the final total price is calculated correctly, including the application of discounts.

```
All tests passed for add_to_cart, remove_from_cart, and checkout methods.
```

Testing Discount Mechanisms

The `test_discount_classes()` function ensures that the *Discount* abstract class enforces the implementation of the `apply_discount()` method in its subclasses. Both *PercentageDiscount* and *FixedAmountDiscount* are tested to confirm that they apply reductions correctly to the total price. Additionally, the function verifies that multiple discounts can be applied sequentially.

```
Confirmed: Discount is abstract and cannot be instantiated directly.
PercentageDiscount.apply_discount works correctly.
FixedAmountDiscount.apply_discount works correctly.
Polymorphic discount application works correctly.
```

All tests pass successfully, demonstrating that the inventory management and user checkout system operates as intended.

Conclusion

During the development of this project, a few challenges arose, particularly in structuring the discount application process and handling multiple product types within the cart. Through careful use of encapsulation, important attributes were protected, preventing unintended modifications. Inheritance allowed for efficient code reuse, with specialized product types extending the base *Product* class. Polymorphism made it possible to apply discounts dynamically, regardless of their type.

Implementing a database for persistent storage would enhance the system by allowing users to save and retrieve their cart contents. A Graphical User Interface could also be introduced to improve user interaction. Additionally, more advanced discounting strategies could be implemented, such as multi-tiered or conditional discounts.