Maman Saly Mbengue

Project report

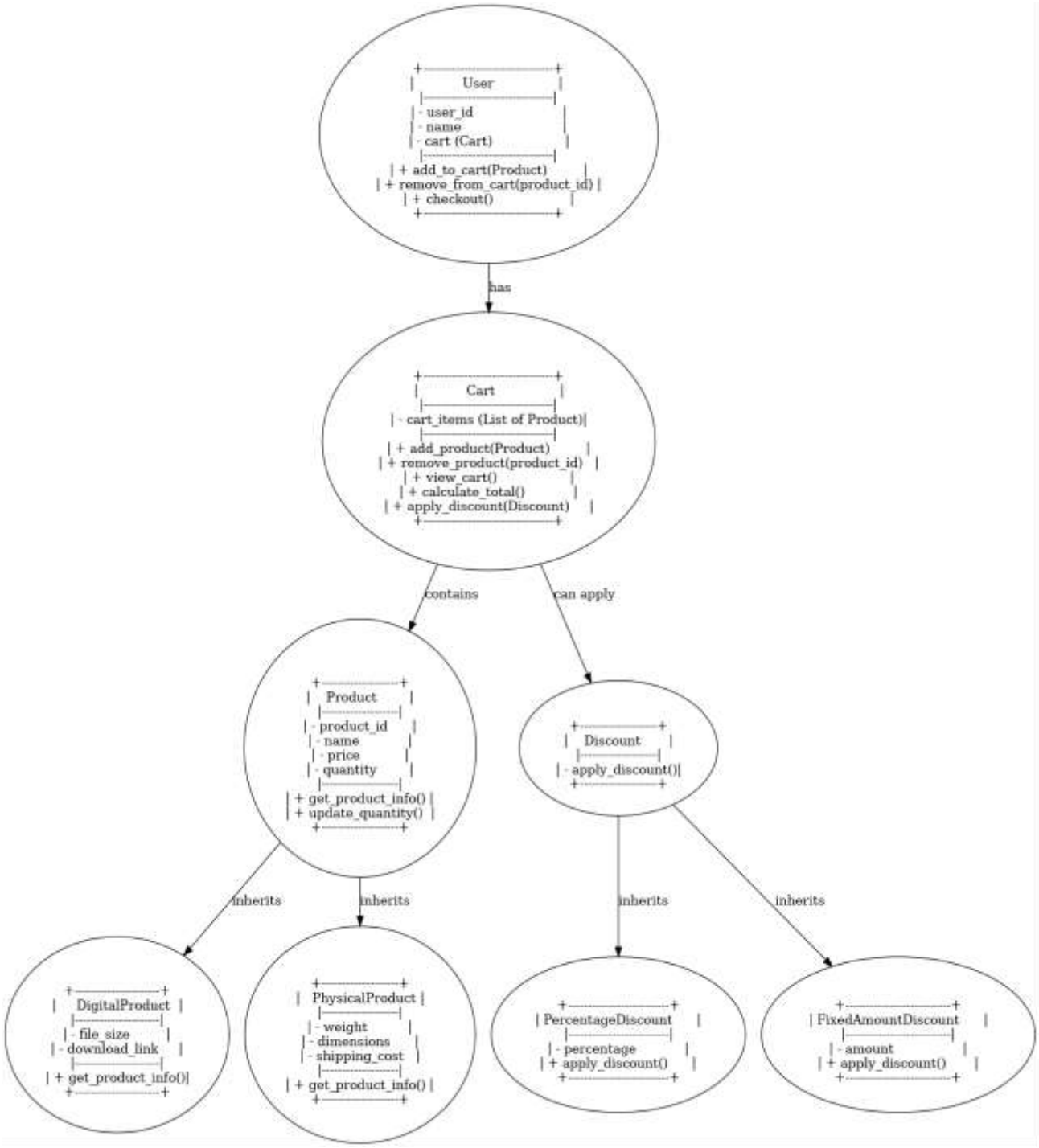
Description:

This project simulates an e-commerce shopping cart system using object-oriented programming (OOP) concepts. The system consists of different types of products, including digital and physical products, and allows users to manage their shopping carts by adding/removing items, applying discounts, and performing checkout operations. The project also includes functionalities for applying different discount types (percentage-based and fixed amount discounts) and managing user operations seamlessly.The system implements various OOP concepts, such as inheritance, polymorphism, encapsulation, and abstraction, providing a structured and modular approach to handling product details and user interactions in a shopping cart.

Instructions:

To use the shopping cart system, first create instances of `DigitalProduct` and `PhysicalProduct` with the required attributes. For digital products, you need to specify details like `file_size` and `download_link`, while for physical products, you must provide attributes such as `weight`, `dimensions`, and `shipping_cost`. Next, create instances of the `User` class, each representing a user with their own shopping cart. Products can be added to a user's cart using the `add_to_cart()` method, and the cart can be viewed using the `view_cart()` method. Users can apply discounts by creating instances of either the `PercentageDiscount` or `FixedAmountDiscount` classes and passing them to the `checkout()` method. The checkout process will calculate the total price, apply the specified discount, and clear the cart after the transaction is completed. Throughout the process, the system ensures that all cart operations are managed through user-friendly methods that hide the underlying complexity. This setup allows users to easily manage their shopping experience, apply discounts, and finalize their purchases.

Structure:


User
- user_id
- name
- cart (Cart)
+ add_to_cart(Product)
+ remove_from_cart(product_id)
+ checkout()
has
Cart
- cart_items (List of Product)
+ add_product(Product)
+ remove_product(product_id)
+ view_cart()
+ calculate_total()
+ apply_discount(Discount)
contains
can apply
Product
- product_id
- name
- price
- quantity
+ get_product_info()
+ update_quantity()
Discount
- apply_discount()
inherits
inherits
inherits
inherits
DigitalProduct
- file_size
- download_link
+ get_product_info()
PhysicalProduct
- weight
- dimensions
- shipping_cost
+ get_product_info()
PercentageDiscount
- percentage
+ apply_discount()
FixedAmountDiscount
- amount
+ apply_discount()

Brief Summary on Developed Classes

- **Product Class**: Represents a product with attributes such as `product_id`, `name`, `price`, and `quantity`. It includes methods like `update_quantity()` to modify stock and `get_product_info()` to retrieve details about the product.

- **DigitalProduct Class**: Inherits from `Product`, adding `file_size` and `download_link` attributes to handle digital items. It overrides `get_product_info()` to display digital-specific details.

- **PhysicalProduct Class**: Also inherits from `Product`, introducing additional attributes like `weight`, `dimensions`, and `shipping_cost` to handle physical goods. It overrides `get_product_info()` to include shipping-related information.

- **Cart Class**: Manages a collection of products in a private list called `cart_items`. It includes methods such as `add_product()`, `remove_product()`, `view_cart()`, `calculate_total()`, and `apply_discount()` to facilitate cart management and discount application.

- **User Class**: Represents a user with a `user_id`, `name`, and a `cart`. It provides methods like `add_to_cart()`, `remove_from_cart()`, and `checkout()` to interact with the shopping process.

- **Discount Class (Abstract)**: Serves as a base class for discount strategies, defining an abstract method `apply_discount()`.

- **PercentageDiscount Class**: Implements `apply_discount()` to apply a percentage-based discount on the cart total.

- **FixedAmountDiscount Class**: Implements `apply_discount()` to reduce the total price by a fixed amount.

Verification of sanity of code:

Scenario 1: Adding Products to User Carts

Use the `add_to_cart` method under the `User` class. This method takes one argument: the product that the user wants to add to their cart. In the following case, I create two users and add digital products to User 1's cart and physical products to User 2's cart.

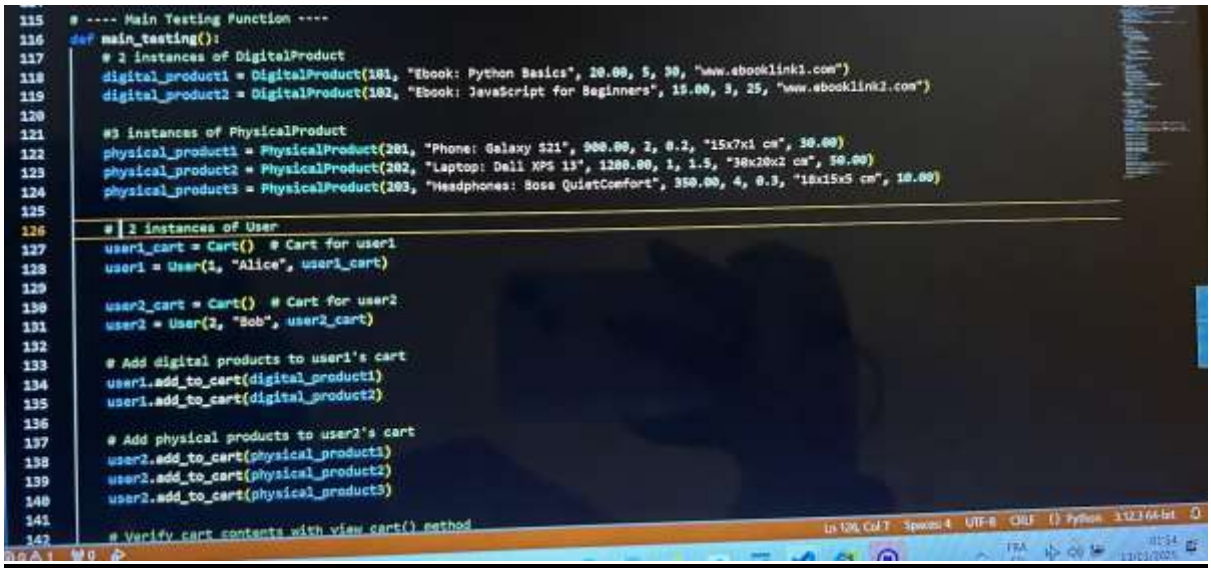
Before Adding Products:
User 1's cart is empty.
User 2's cart is empty.

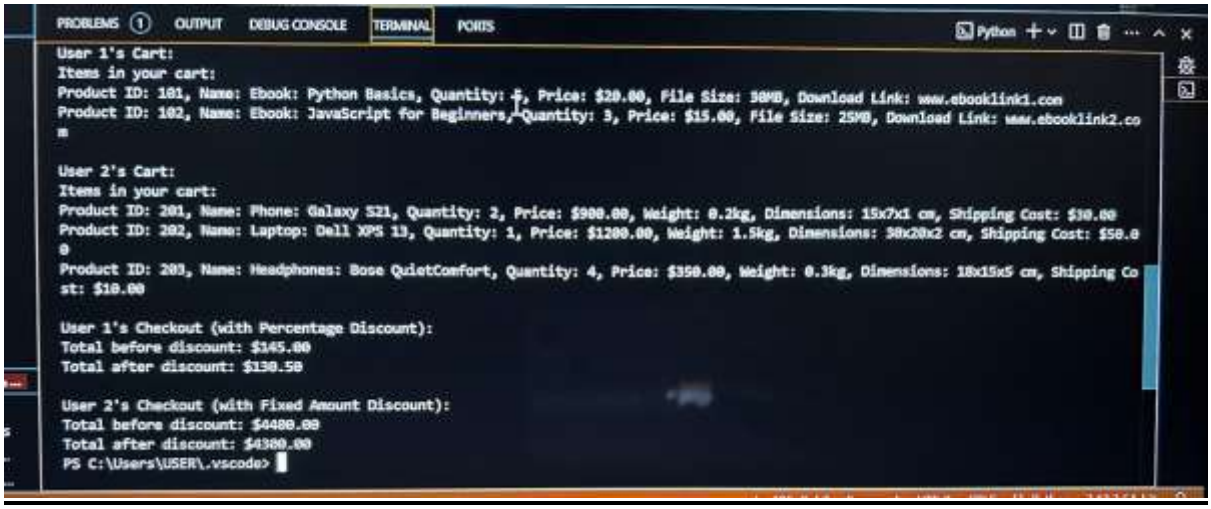
After Adding Products:

User 1's cart contains 2 digital products.

User 2's cart contains 3 physical products.

Code Execution:



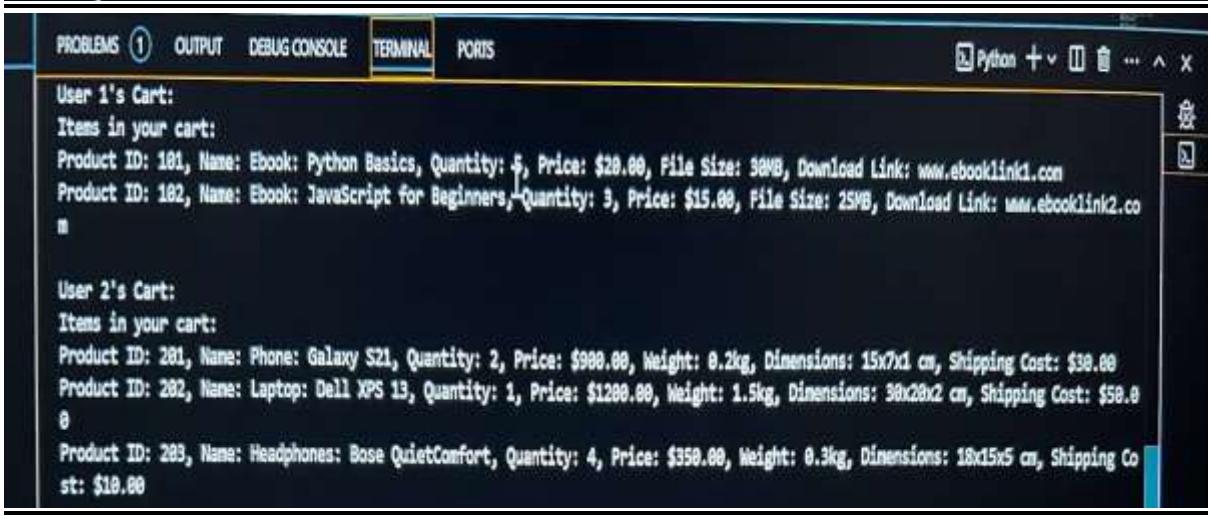


Scenario 2: Viewing Cart Contents

Use the `view_cart` method under the `Cart` class. This method prints out the details of all products in a user's cart.

Code Execution:

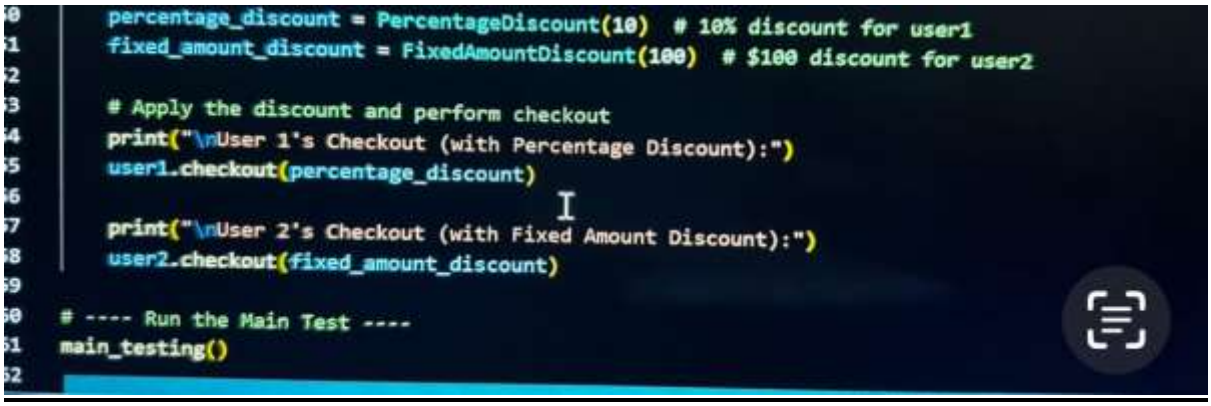
```
142    # Verify cart contents with view_cart() method
143    print("\nUser 1's Cart:")
144    user1.cart.view_cart()
145
146    print("\nUser 2's Cart:")
147    user2.cart.view_cart()
148
149    # Create instances of discounts
150    percentage_discount = PercentageDiscount(10)  # 10% discount for user1
```



Scenario 3: Applying Discounts and Checking Out

Use the `checkout` method under the `User` class. This method calculates the total cart value and applies discounts before clearing the cart.

Code Execution:



```
    percentage_discount = PercentageDiscount(10)  # 10% discount for user1
    fixed_amount_discount = FixedAmountDiscount(100)  # $100 discount for user2

    # Apply the discount and perform checkout
    print("\nUser 1's Checkout (with Percentage Discount):")
    user1.checkout(percentage_discount)

    print("\nUser 2's Checkout (with Fixed Amount Discount):")
    user2.checkout(fixed_amount_discount)

# ---- Run the Main Test ----
main_testing()
```

```
st: $10.00

User 1's Checkout (with Percentage Discount):
Total before discount: $145.00
Total after discount: $130.50

User 2's Checkout (with Fixed Amount Discount):
Total before discount: $4400.00
Total after discount: $4300.00
PS C:\Users\USER\.vscode>
```

Conclusion:

Working on this project came with a few challenges, but it was a great learning experience. One of the biggest difficulties was making sure all the classes worked well together while keeping the code organized and easy to understand. Figuring out how to connect the `User`, `Cart`, and `Product` classes took some time, especially when making sure products could be added and removed properly. Another tricky

part was applying discounts in a way that was flexible but still simple to use. Creating an abstract `Discount` class and having `PercentageDiscount` and `FixedAmountDiscount` inherit from it helped make the code more reusable. Keeping track of product quantities was another challenge, as we had to make sure users couldn't buy more than what was available. Despite these challenges, using object-oriented programming made the project much easier to manage. Breaking everything into separate classes helped keep the code clean and organized. However, there are some areas that could be improved. For example, adding a way to save user carts and product inventory so they don't reset every time the program runs would make the system more useful. It would also be interesting to add more types of discounts or special deals. Overall, this project was a great way to practice using classes and objects in Python, and it really showed how important good code structure is in making a program easy to use and expand in the future.