Final Exam Python

Ali ZAINOUL
ali.zainoul.az@gmail.com

August 2, 2024

Class Diagram

You were tasked to implement an Online Shopping System in Python, given the following class diagram:

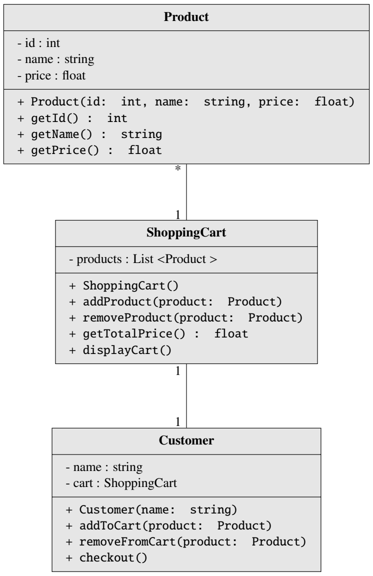

Figure 1: Class diagram for the Online Shopping System

Analysis of the Class Diagram

The class diagram represents an Online Shopping System with three classes: `Product`, `ShoppingCart`, and `Customer`. Let's analyze each class:

`Product`

- **Attributes:**
  - `id: int` (private)
  - `name: string` (private)
  - `price: float` (private)
- **Methods:**
  - `Product(id: int, name: string, price: float)` (public constructor)
  - `getId(): int` (public method)
  - `getName(): string` (public method)
  - `getPrice(): float` (public method)

`ShoppingCart`

- **Attributes:**
  - `products: List<Product>` (private)
- **Methods:**
  - `ShoppingCart()` (public constructor)
  - `addProduct(product: Product)` (public method)
  - `removeProduct(product: Product)` (public method)
  - `getTotalPrice(): float` (public method)
  - `displayCart()` (public method)

`Customer`

- **Attributes:**
  - `name: string` (private)
  - `cart: ShoppingCart` (private)
- **Methods:**
  - `Customer(name: string)` (public constructor)
  - `addToCart(product: Product)` (public method)
  - `removeFromCart(product: Product)` (public method)
  - `checkout()` (public method)

The class diagram shows the relationships between the classes:

- The `Customer` class has a composition relationship with the `ShoppingCart` class (1:1). A `Customer` has one `ShoppingCart`, and the `ShoppingCart` is owned by the `Customer`.
- The `ShoppingCart` class has an aggregation relationship with the `Product` class (1 to *). A `ShoppingCart` can contain multiple `Products`, but the `Products` exist independently of the `ShoppingCart`.

In summary, the class diagram illustrates the structure of an Online Shopping System, where `Customers` can have their own `ShoppingCart` and interact with `Products` through operations such as adding, removing, and checking out items.

Class Product Implementation

```
# File: Product.py
class Product:
    def __init__(self, id: int, name: str, price: float):
        """
        Constructor for the Product class.

        Parameters:
            id (int): The product ID.
            name (str): The product name.
            price (float): The product price.
        """
        self.__id = id  # Private attribute to store the product ID
        self.__name = name  # Private attribute to store the product name
        self.__price = price  # Private attribute to store the product price

    def get_id(self) -> int:
        """
        Get the product ID.

        Returns:
            int: The product ID.
        """
        return self.__id

    def get_name(self) -> str:
        """
        Get the product name.

        Returns:
            str: The product name.
        """
        return self.__name

    def get_price(self) -> float:
        """
        Get the product price.

        Returns:
            float: The product price.
        """
        return self.__price
```

In this code, we use double underscores (__) to declare **private attributes** (__id, __name, and __price). The constructor **__init__** takes the arguments id, name, and price and initializes the private attributes. The methods get_id, get_name, and get_price are **public methods** that allow **accessing the private attributes** by returning their values.

Good luck! ♥