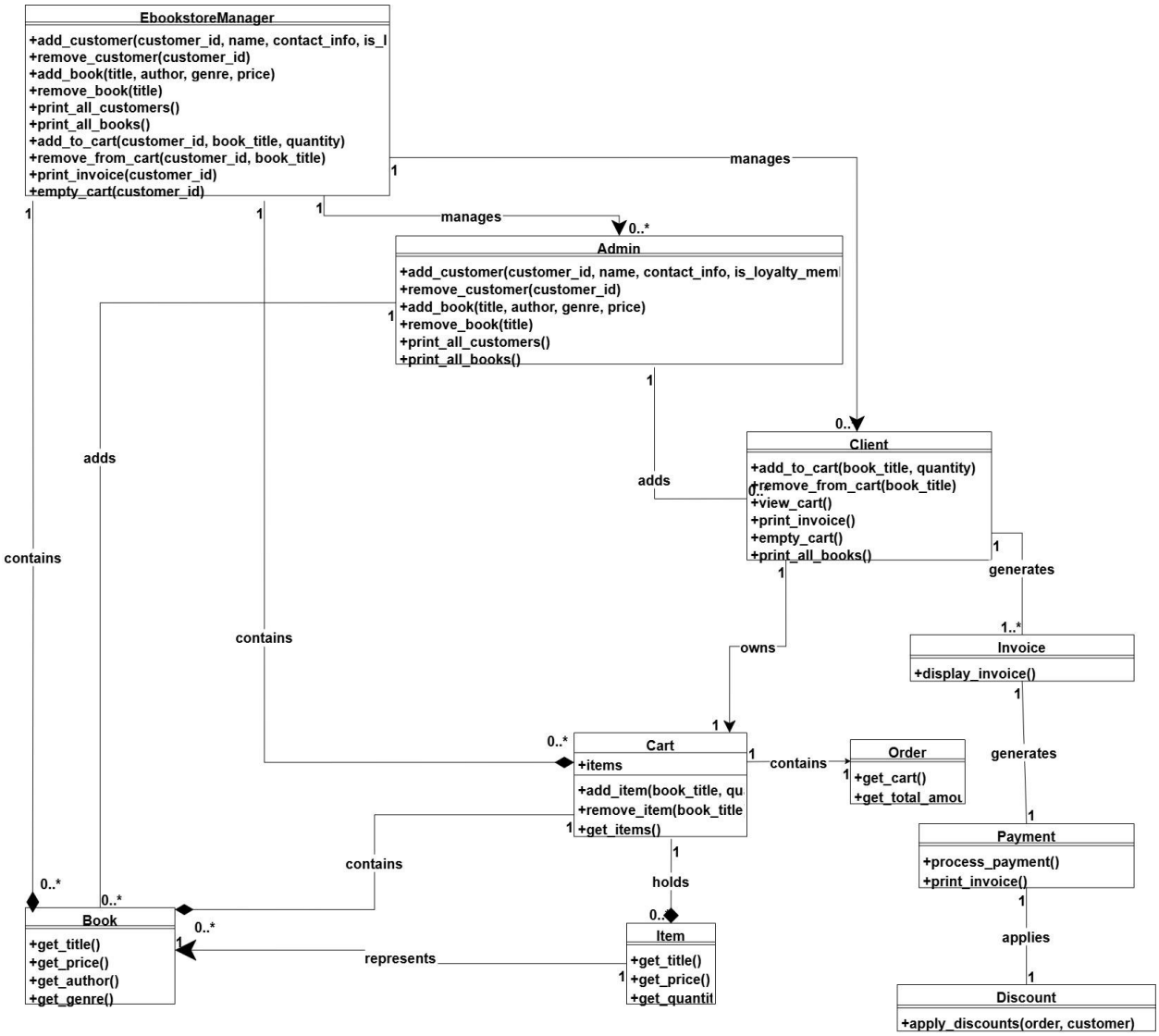
Naser Mohamed

Program. Fund.

202215515

Uml diagram


EbookstoreManager
+add_customer(customer_id, name, contact_info, is_l
+remove_customer(customer_id)
+add_book(title, author, genre, price)
+remove_book(title)
+print_all_customers()
+print_all_books()
+add_to_cart(customer_id, book_title, quantity)
+remove_from_cart(customer_id, book_title)
+print_invoice(customer_id)
+empty_cart(customer_id)
manages
manages
Admin
+add_customer(customer_id, name, contact_info, is_loyalty_mem
+remove_customer(customer_id)
+add_book(title, author, genre, price)
+remove_book(title)
+print_all_customers()
+print_all_books()
adds
adds
Client
+add_to_cart(book_title, quantity)
+remove_from_cart(book_title)
+view_cart()
+print_invoice()
+empty_cart()
+print_all_books()
generates
contains
contains
owns
Invoice
+display_invoice()
Cart
+items
+add_item(book_title, qu
+remove_item(book_title
+get_items()
contains
Order
+get_cart()
+get_total_amou
generates
contains
holds
Payment
+process_payment()
+print_invoice()
Book
+get_title()
+get_price()
+get_author()
+get_genre()
represents
Item
+get_title()
+get_price()
+get_quantit
applies
Discount
+apply_discounts(order, customer)

Explanation

Relationship types for this UML diagram may be distinguished by aggregation and association. A diamond sign gives aggregation; it denotes the "whole-part" relationship where the whole may exist independently of its parts. Here, EbookstoreManager aggregates books, carts, and items because a manager maintains a collection of these objects, though they can also be independently created and used outside the manager's instance. Example: A cart is an aggregate of Items because a cart consists of many items, but the items could have existed independently of the Cart. Similarly, EbookstoreManager is an aggregate of carts because the manager manages many carts but only owns them in a restrictive sense that would prevent them from existing outside the manager's world. Association shows a direct relationship between classes without ownership. Associations: EbookstoreManager - Admin is, say, one manager can have several admins. The admins could operate things, for example, adding or removing customers or books, but the admin does not 'belong' intrinsically to the manager. Again, Client- Cart; every client owns a cart, but a cart should not be independent of the client. Other associations are Invoice and Payment, where the Invoice is generated and paid by Payment, and the Order is associated with a Cart, stating that orders contain carts. These associations imply interactions but not ownership or life-cycle dependency between the objects.

Code

```python
# book.py

class Book:
    """
    Represents an e-book with details such as title, author, genre, and price.
    """
    def __init__(self, title, author, genre, price):
        self.title = title
        self.author = author
        self.genre = genre
        self.price = price

    def get_title(self):
        return self.title

    def get_author(self):
        return self.author

    def get_genre(self):
        return self.genre

    def get_price(self):
        return self.price

    def set_price(self, new_price):
        self.price = new_price
```

```python
# cart.py

class Cart:
    """
    Manages items added by the customer, with each item’s quantity.
    """
    def __init__(self):
        self.items = {}

    def add_to_cart(self, book, quantity):
        if book in self.items:
            self.items[book] += quantity
        else:
```

```
            self.items[book] = quantity

    def remove_from_cart(self, book):
        if book in self.items:
            del self.items[book]

    def update_quantity(self, book, new_quantity):
        if book in self.items:
            self.items[book] = new_quantity

    def get_cart_items(self):
        return self.items
```

```
from ebookstore_manager import EbookstoreManager

class Client:
    """
    Represents a customer (client) of the e-bookstore who can browse, add items
to cart, and purchase books.
    """
    def __init__(self, manager, customer_id):
        self.manager = manager
        self.customer_id = customer_id

    def add_to_cart(self, book_title, quantity):
        """
        Customer adds a specified quantity of a book to their cart.
        """
        self.manager.add_to_cart(self.customer_id, book_title, quantity)

    def remove_from_cart(self, book_title):
        """
        Customer removes a book from their cart.
        """
        self.manager.remove_from_cart(self.customer_id, book_title)

    def view_cart(self):
        """
        Displays the contents of the customer's cart in an organized table format
using padding.
        """
        customer = self.manager.customers.get(self.customer_id)
```

```
        if customer:
            # Print header
            print(f"{'Book Title':<30} {'Quantity':<10} {'Price per Unit':<15}
{'Total Price':<15}")
            print("-" * 70)  # Separator line

            # Loop through the cart items and display them in a formatted way
            for item, qty in customer.get_cart().items.items():
                title = item.get_title()
                price = item.get_price()
                total_price = price * qty
                print(f"{title:<30} {qty:<10} ${price:<14.2f}
${total_price:<14.2f}")
        else:
            print("Customer not found.")


    def print_invoice(self):
        """
        Customer prints the invoice for their current cart.
        """
        self.manager.print_invoice(self.customer_id)

    def empty_cart(self):
        """
        Customer empties their cart.
        """
        self.manager.empty_cart(self.customer_id)


    def print_all_books(self):
        """
        Customer print all books
        """
        self.manager.print_all_books()
```

```python
from cart import Cart

class Customer:
    """
    Represents a customer in the e-bookstore system.
    """
    def __init__(self, customer_id, name, contact_info, is_loyalty_member=False):
        self.customer_id = customer_id
        self.name = name
        self.contact_info = contact_info
        self.is_loyalty_member = is_loyalty_member
        self.cart = Cart()

    def get_customer_id(self):
        return self.customer_id

    def get_name(self):
        return self.name

    def get_contact_info(self):
        return self.contact_info

    def set_contact_info(self, new_info):
        self.contact_info = new_info

    def is_loyal_member(self):
        return self.is_loyalty_member

    def get_cart(self):
        return self.cart

    def set_cart(self, new_cart):
        self.cart = new_cart
```

```python
# discount.py

class Discount:
    """
    Calculates applicable discounts based on loyalty status and bulk purchases.
    """
    LOYALTY_DISCOUNT_RATE = 0.10
    BULK_DISCOUNT_RATE = 0.20
    BULK_THRESHOLD = 5

    def calculate_loyalty_discount(self, order_total):
        return order_total * Discount.LOYALTY_DISCOUNT_RATE

    def calculate_bulk_discount(self, total_quantity, order_total):
        if total_quantity >= Discount.BULK_THRESHOLD:
            return order_total * Discount.BULK_DISCOUNT_RATE
        return 0

    def apply_discounts(self, order,custo):
        total_discount = 0
        if order.total_quantity >= Discount.BULK_THRESHOLD:
            total_discount +=
self.calculate_bulk_discount(order.get_total_quantity(), order.get_order_total())
        if custo.is_loyal_member():
            total_discount +=
self.calculate_loyalty_discount(order.get_order_total())
        return total_discount
```

```python
from customer import Customer
from book import Book
from order import Order
from payment import Payment
from discount import Discount
from cart import Cart

class EbookstoreManager:
    """
    Central class to manage all functionalities of the e-bookstore,
    including customer management, cart operations, discount application,
    and invoice generation.
    """
    def __init__(self):
        self.customers = {}
        self.books = {}
        self.discount = Discount()

    # Customer management
    def add_customer(self, customer_id, name, contact_info,
is_loyalty_member=False):
        """
        Adds a new customer to the system with a unique cart.
        """
        customer = Customer(customer_id, name, contact_info, is_loyalty_member)
        self.customers[customer_id] = customer
        print(f"Customer '{name}' with ID '{customer_id}' added successfully.")

    def remove_customer(self, customer_id):
        """
        Removes an existing customer from the system by ID.
        """
        if customer_id in self.customers:
            del self.customers[customer_id]
            print(f"Customer with ID '{customer_id}' removed successfully.")
        else:
            print(f"Customer with ID '{customer_id}' not found.")

    # Book management
    def add_book(self, title, author, genre, price):
        """
        Adds a new book to the store's catalog.
        """
        book = Book(title, author, genre, price)
        self.books[title] = book
```

```
        print(f"Book '{title}' added to the catalog.")

    def remove_book(self, title):
        """
        Removes a book from the store's catalog.
        """
        if title in self.books:
            del self.books[title]
            print(f"Book '{title}' removed from the catalog.")
        else:
            print(f"Book '{title}' not found in the catalog.")

    # Shopping cart operations
    def add_to_cart(self, customer_id, book_title, quantity):
        """
        Adds a specified quantity of a book to a customer's cart.
        """
        customer = self.customers.get(customer_id)
        book = self.books.get(book_title)
        if customer and book:
            customer.get_cart().add_to_cart(book, quantity)
            print(f"Added {quantity} of '{book_title}' to {customer.name}'s
cart.")
        else:
            print("Customer or book not found.")

    def remove_from_cart(self, customer_id, book_title):
        """
        Removes a book from a customer's cart.
        """
        customer = self.customers.get(customer_id)
        book = self.books.get(book_title)
        if customer and book:
            customer.get_cart().remove_from_cart(book)
            print(f"Removed '{book_title}' from {customer.name}'s cart.")
        else:
            print("Customer or book not found.")

    def empty_cart(self, customer_id):
        """
        Empties the cart of the specified customer.
        """
        customer = self.customers.get(customer_id)
        if customer:
            customer.set_cart(Cart())  # Assign a new empty cart to the customer
```

```
            print(f"{customer.name}'s cart has been emptied.")
        else:
            print(f"Customer with ID '{customer_id}' not found.")

    # Invoice generation
    def print_invoice(self, customer_id):
        """
        Prints the invoice for a customer's current cart, applying discounts and
displaying totals.
        """
        customer = self.customers.get(customer_id)
        if customer:
            # Create an order based on the customer's cart
            order = Order(customer.get_cart())

            # Apply discounts
            discount_amount = self.discount.apply_discounts(order,customer)

            # Process payment and generate invoice
            payment = Payment(order, self.discount,customer)
            payment.process_payment()

            # Print formatted invoice
            payment.print_invoice()

            # Optionally empty the cart after generating the invoice
            self.empty_cart(customer_id)
        else:
            print(f"Customer with ID '{customer_id}' not found.")


    def print_all_customers(self):
        """
        Prints details of all customers in the system in an organized table
format.
        """
        if not self.customers:
            print("No customers found.")
        else:
            # Print header
            print("\n" + "*" * 100)
            print(f"{'All Customers:':^100}")
            print("*" * 100)
            print(f"{'Customer ID':<20} {'Name':<30} {'Contact Info':<30}
{'Loyalty Member':<15}")
```

```
            print("-" * 100)

            # Loop through customers and display them in a formatted way
            for customer in self.customers.values():
                print(f"{customer.customer_id:<20} {customer.name:<30}
{customer.contact_info:<30} {str(customer.is_loyalty_member):<15}")

            print("-" * 100)
            print("\n")


    def print_all_books(self):
        """
        Prints details of all books in the store's catalog in an organized table
format.
        """
        if not self.books:
            print("No books found in the catalog.")
        else:
            # Print header
            print("\n" + "*" * 100)
            print(f"{'All Books in Catalog:':^100}")
            print("*" * 100)
            print(f"{'Title':<30} {'Author':<25} {'Genre':<20} {'Price':<15}")
            print("-" * 100)

            # Loop through books and display them in a formatted way
            for book in self.books.values():
                print(f"{book.title:<30} {book.author:<25} {book.genre:<20}
${book.price:<14.2f}")

            print("-" * 100)
            print("\n")
```

```python
# invoice.py

class Invoice:
    """
    Generates a formatted invoice for the customer's order, including VAT and
discounts.
    """
    VAT_RATE = 0.08

    def __init__(self, order, discount_amount):
        self.order = order
        self.discount_amount = discount_amount
        self.vat_amount = self.calculate_vat()
        self.final_amount = self.calculate_final_amount()

    def calculate_vat(self):
        return (self.order.get_order_total() - self.discount_amount) *
Invoice.VAT_RATE

    def calculate_final_amount(self):
        return self.order.get_order_total() - self.discount_amount +
self.vat_amount

    def display_invoice(self):
        # Print header
        print("\n" + "="*50)
        print(f"{'Item':<20}{'Quantity':<10}{'Unit Price':<10}{'Total
Price':<10}")
        print("-"*50)

        # Print each item
        for book, quantity in self.order.order_items.items():
            total_price = book.get_price() * quantity
            print(f"{book.get_title():<20}{quantity:<10}{book.get_price():<10.2f}
{total_price:<10.2f}")

        # Print footer with discounts, VAT, and final total
        print("-"*50)
        print(f"{'Subtotal':<40}{self.order.get_order_total():<10.2f}")
        print(f"{'Discount':<40}-{self.discount_amount:<10.2f}")
        print(f"{'VAT (8%)':<40}+{self.vat_amount:<10.2f}")
        print("="*50)
        print(f"{'Total':<40}{self.final_amount:<10.2f}")
        print("="*50 + "\n")
```

```python
# order.py

class Order:
    """
    Represents a customer's order, which includes items and quantities.
    """
    def __init__(self, cart):
        self.order_items = cart.get_cart_items()
        self.total_quantity = sum(self.order_items.values())

    def get_order_total(self):
        return sum(book.get_price() * qty for book, qty in
self.order_items.items())

    def get_total_quantity(self):
        return self.total_quantity
```

```python
# payment.py

from invoice import Invoice

class Payment:
    """
    Processes payment by applying discounts, calculating VAT, and generating an
invoice.
    """
    def __init__(self, order, discount,custo):
        self.order = order
        self.discount = discount
        self.custo = custo
        self.invoice = None

    def process_payment(self):
        discount_amount = self.discount.apply_discounts(self.order,self.custo)
        self.invoice = Invoice(self.order, discount_amount)

    def print_invoice(self):
        self.invoice.display_invoice()
```

```python
from ebookstore_manager import EbookstoreManager

class Admin:
    """
    Represents an administrator who can manage customers and books in the e-bookstore.
    """
    def __init__(self, manager):
        self.manager = manager

    def add_customer(self, customer_id, name, contact_info, is_loyalty_member=False):
        """
        Admin adds a customer to the system.
        """
        self.manager.add_customer(customer_id, name, contact_info, is_loyalty_member)

    def remove_customer(self, customer_id):
        """
        Admin removes a customer from the system.
        """
        self.manager.remove_customer(customer_id)

    def add_book(self, title, author, genre, price):
        """
        Admin adds a book to the catalog.
        """
        self.manager.add_book(title, author, genre, price)

    def remove_book(self, title):
        """
        Admin removes a book from the catalog.
        """
        self.manager.remove_book(title)

    def print_all_customers(self):
        """
        Admin removes a book from the catalog.
        """
        self.manager.print_all_customers()

    def print_all_books(self):
        """
        Admin removes a book from the catalog.
```

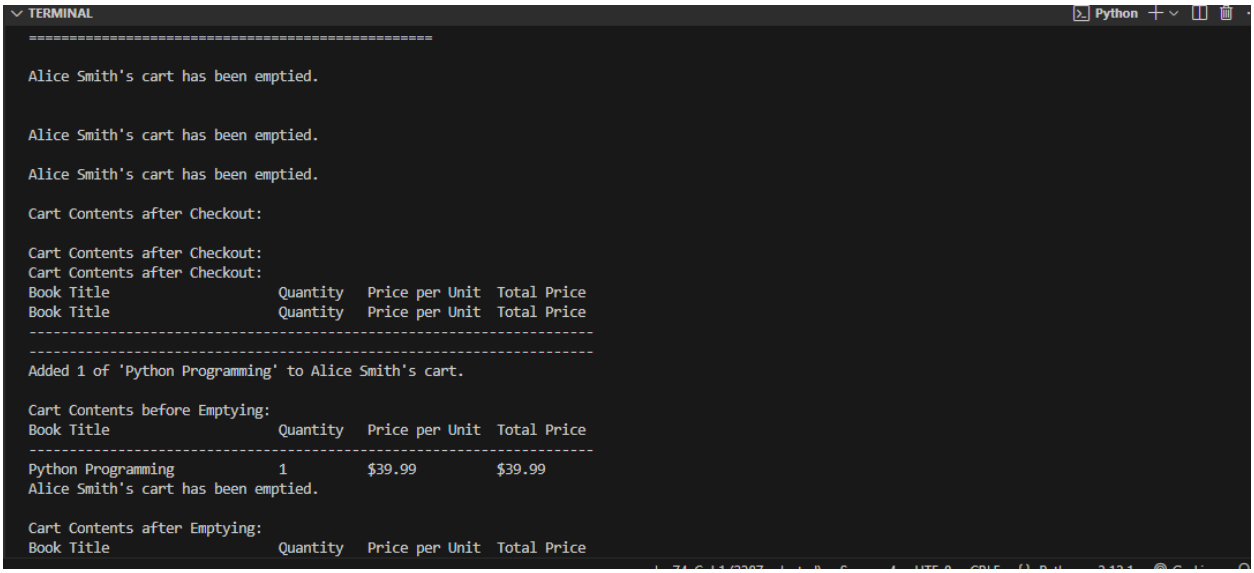
```
        """
        self.manager.print_all_books()
```

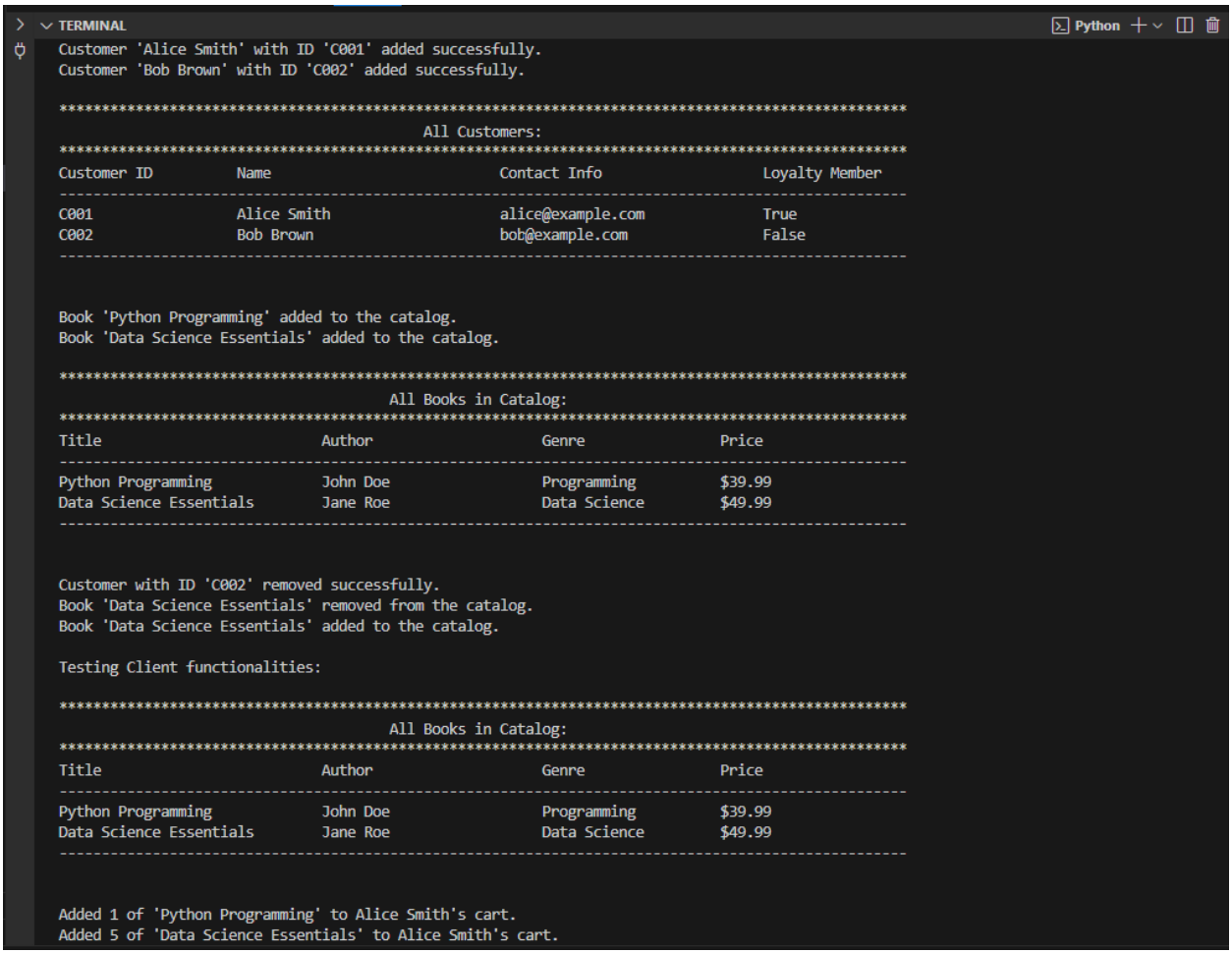
Test File

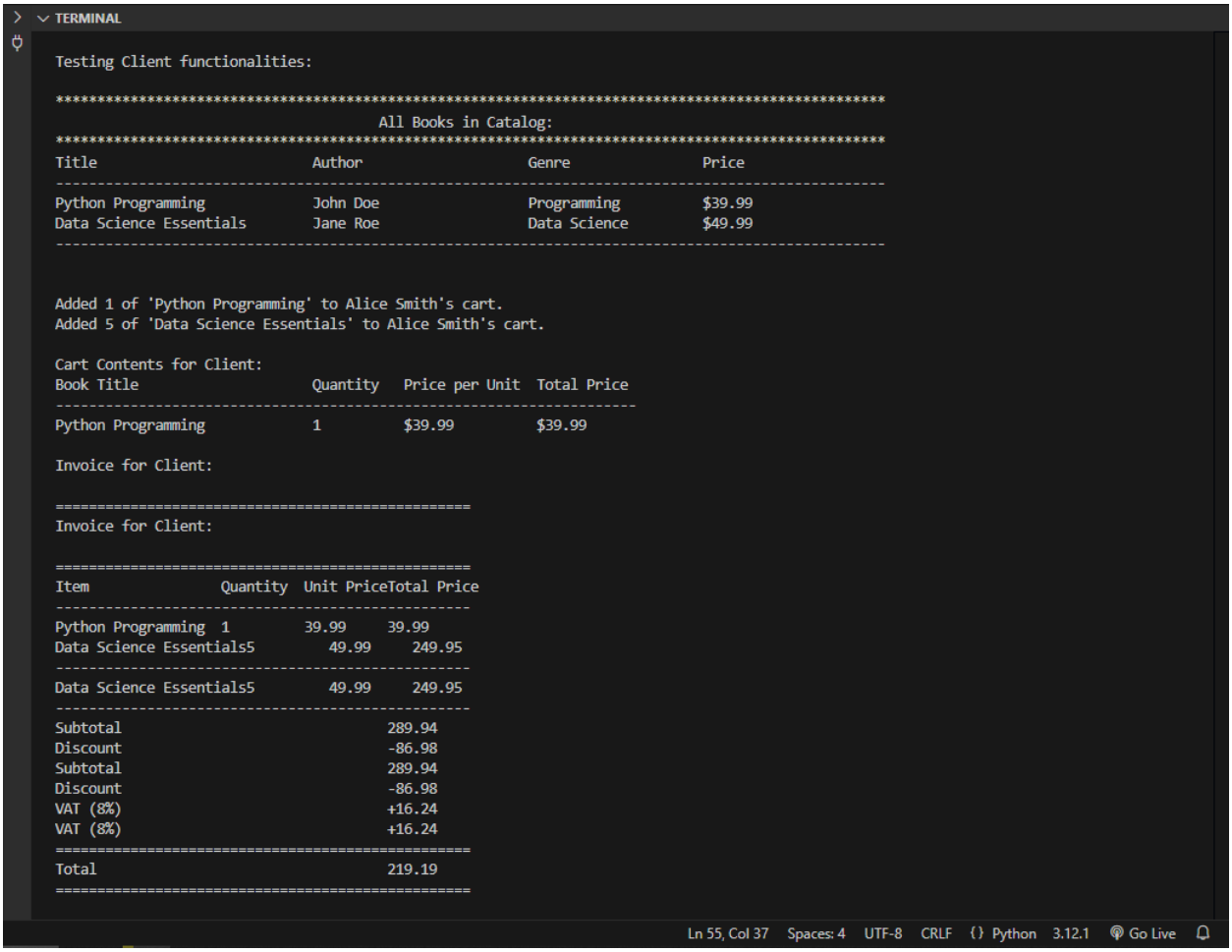
```
# test_ebookstore.py

from ebookstore_manager import EbookstoreManager
from admin import Admin
from client import Client

def test_ebookstore():
    # Step 1: Initialize the EbookstoreManager
    manager = EbookstoreManager()

    # Step 2: Create an Admin instance to manage customers and books
    admin = Admin(manager)

    # Step 3: Test Admin functionalities
    print("Testing Admin functionalities:")

    # Add a customer
    admin.add_customer(customer_id="C001", name="Alice Smith",
contact_info="alice@example.com", is_loyalty_member=True)
    admin.add_customer(customer_id="C002", name="Bob Brown",
contact_info="bob@example.com", is_loyalty_member=False)

    # display all customer
    admin.print_all_customers()


    # Add books to the catalog
    admin.add_book(title="Python Programming", author="John Doe",
genre="Programming", price=39.99)
    admin.add_book(title="Data Science Essentials", author="Jane Roe",
genre="Data Science", price=49.99)


    admin.print_all_books()

    # Remove a customer and book
```

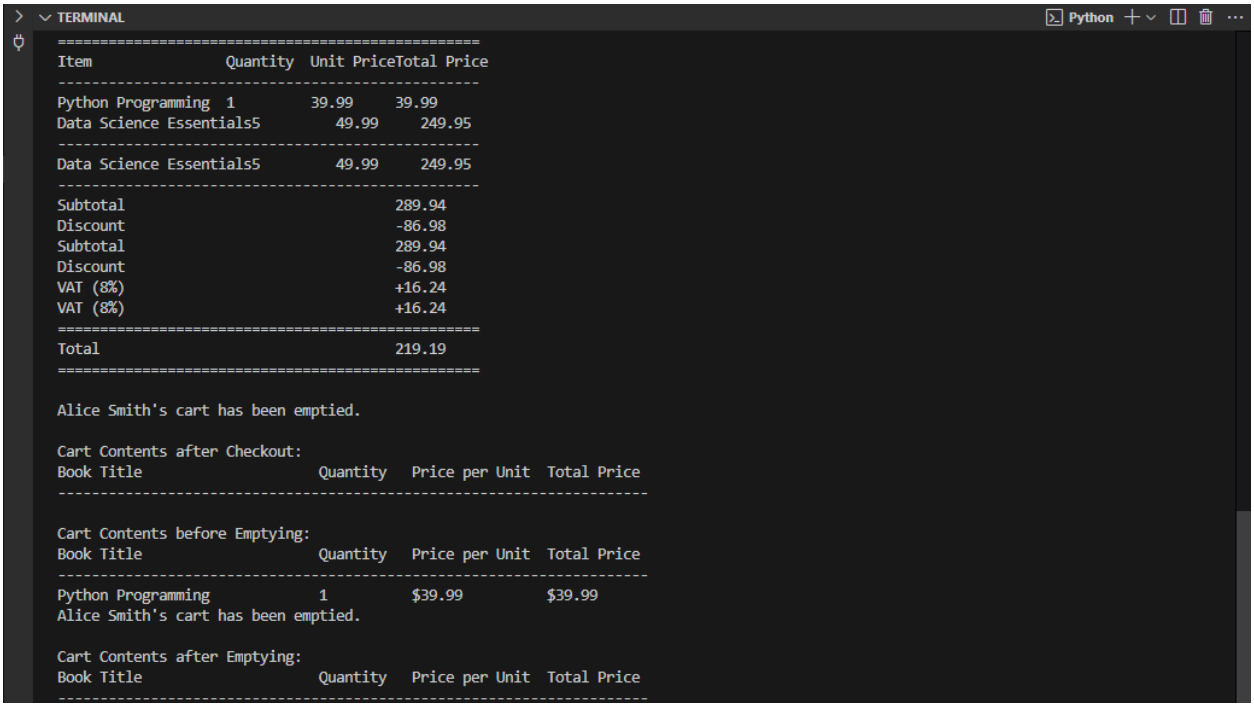
```
    admin.remove_customer("C002")
    admin.remove_book("Data Science Essentials")

    #add again the book for testing
    admin.add_book(title="Data Science Essentials", author="Jane Roe",
genre="Data Science", price=49.99)

    # Step 4: Create a Client instance for the added customer
    client = Client(manager, customer_id="C001")

    # Step 5: Test Client functionalities
    print("\nTesting Client functionalities:")

    client.print_all_books()

    # Add books to the cart
    client.add_to_cart("Python Programming", 1)
    client.add_to_cart("Data Science Essentials", 5)

    # View cart to verify items
    print("\nCart Contents for Client:")
    client.view_cart()

    # Generate and print the invoice
    print("\nInvoice for Client:")
    client.print_invoice()

    # View cart after checkout to ensure it's empty
    print("\nCart Contents after Checkout:")
    client.view_cart()

    # Add another item and test cart clearing functionality
    client.add_to_cart("Python Programming", 1)
    print("\nCart Contents before Emptying:")
    client.view_cart()

    client.empty_cart()
    print("\nCart Contents after Emptying:")
    client.view_cart()

if __name__ == "__main__":
    test_ebookstore()
```

Test Results




TERMINAL
Python
Customer 'Alice Smith' with ID 'C001' added successfully.
Customer 'Bob Brown' with ID 'C002' added successfully.
All Customers:
Customer ID Name Contact Info Loyalty Member
C001 Alice Smith alice@example.com True
C002 Bob Brown bob@example.com False
Book 'Python Programming' added to the catalog.
Book 'Data Science Essentials' added to the catalog.
All Books in Catalog:
Title Author Genre Price
Python Programming John Doe Programming $39.99
Data Science Essentials Jane Roe Data Science $49.99
Customer with ID 'C002' removed successfully.
Book 'Data Science Essentials' removed from the catalog.
Book 'Data Science Essentials' added to the catalog.
Testing Client functionalities:
All Books in Catalog:
Title Author Genre Price
Python Programming John Doe Programming $39.99
Data Science Essentials Jane Roe Data Science $49.99
Added 1 of 'Python Programming' to Alice Smith's cart.
Added 5 of 'Data Science Essentials' to Alice Smith's cart.


TERMINAL
Testing Client functionalities:
All Books in Catalog:
Title Author Genre Price
Python Programming John Doe Programming $39.99
Data Science Essentials Jane Roe Data Science $49.99
Added 1 of 'Python Programming' to Alice Smith's cart.
Added 5 of 'Data Science Essentials' to Alice Smith's cart.
Cart Contents for Client:
Book Title Quantity Price per Unit Total Price
Python Programming 1 $39.99 $39.99
Invoice for Client:
Invoice for Client:
Item Quantity Unit PriceTotal Price
Python Programming 1 39.99 39.99
Data Science Essentials5 49.99 249.95
Data Science Essentials5 49.99 249.95
Subtotal 289.94
Discount -86.98
Subtotal 289.94
Discount -86.98
VAT (8%) +16.24
VAT (8%) +16.24
Total 219.19
Ln 55, Col 37 Spaces: 4 UTF-8 CRLF Python 3.12.1 Go Live


TERMINAL
Python
Item Quantity Unit PriceTotal Price
Python Programming 1 39.99 39.99
Data Science Essentials5 49.99 249.95
Data Science Essentials5 49.99 249.95
Subtotal 289.94
Discount -86.98
Subtotal 289.94
Discount -86.98
VAT (8%) +16.24
VAT (8%) +16.24
Total 219.19
Alice Smith's cart has been emptied.
Cart Contents after Checkout:
Book Title Quantity Price per Unit Total Price
Cart Contents before Emptying:
Book Title Quantity Price per Unit Total Price
Python Programming 1 $39.99 $39.99
Alice Smith's cart has been emptied.
Cart Contents after Emptying:
Book Title Quantity Price per Unit Total Price

Summery Learning

In the project, I learned how to implement and manage an e-bookstore system by creating different classes for customers, books, and orders. I studied object-oriented programming principles like encapsulation, inheritance, and polymorphism while designing the system. The project developed admin and client functionality such as adding/removing books and customers, use of the shopping cart, and generation of invoices. In addition, I learned how to handle complex interactions between classes due to aggregation and composition, and besides testing the system against a variety of scenarios to ensure that the functionality is robust and that all requirements are met. In this way, I improved my understanding of class design, testing practices, and real-world application development.

Gethub

https://github.com/NaserM7/Naser-MK/tree/main