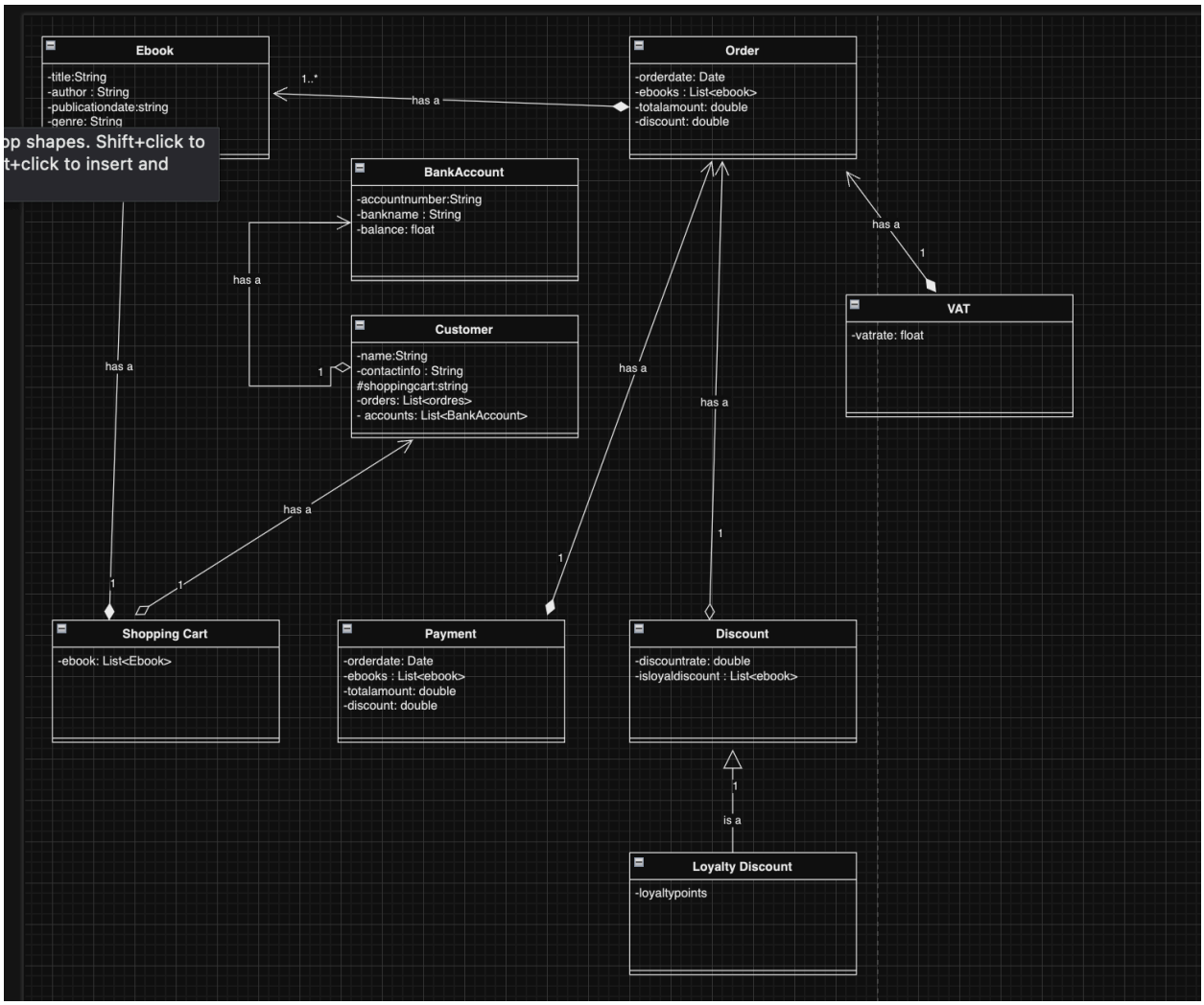
ICS220 - Assignment 2
Mohammad Abdullah Hashim 202107285

- **Design UML Class Diagram Design UML class diagram representing the concepts and relationships in the scenario. Ensure to use inheritance or association (aggregation and composition) relationships. You may make assumptions about attributes (with proper access specified) and concepts not explicitly mentioned in the problem statement. A clear description of the relationships, modularity, and assumptions must be included. All classes to meet the requirements must be identified (a minimum number of required classes is six (6)), and students are expected to demonstrate knowledge of different types of relationships.**



- ## Modularity

Ebook:

This class represents each e-book in the store, with attributes like title, author, publication date, genre, and price. It encapsulates all information related to a single e-book and can be easily expanded as needed to include more details.

Customer:

This class manages customer-specific information, including name, contact information, and associated bank accounts and orders. It serves as the primary representation of a customer within the system and aggregates relevant details about the customer's financial information and purchase history.

ShoppingCart:

This class manages the items a customer has selected for purchase, holding a list of e-books. It provides methods for adding and removing e-books and calculates the total price of items in the cart. The shopping cart is unique to each customer and essential for the order creation process.

Order:

This class is responsible for managing the final order details after a customer completes their purchase. It includes attributes for the order date, list of e-books, total amount, and any discounts applied. The Order class plays a central role in the checkout process and maintains information about what the customer has purchased.

Discount:

This class represents discounts that can be applied to an order. It contains a discount rate and determines if a loyalty discount applies. A subclass, LoyaltyDiscount, inherits from Discount to provide additional discounts for loyalty program members, allowing flexible and customizable discounts.

Payment:

This class processes the payment for an order, including the order date, list of e-books, total amount, and discount details. It is closely related to the Order class, as payment is triggered once an order is finalized. This class could be expanded to handle different payment methods and statuses if needed.

VAT:

This class calculates VAT at a fixed rate and applies it to the total amount in an order. It allows the system to add tax information separately from other pricing calculations, making VAT easy to adjust as needed.

BankAccount:

This class represents a customer's bank account information, including account number, bank name, and balance. It allows linking financial information to the customer and provides flexibility if the system needs to track multiple accounts per customer.

- ## Description: Relationships

Relationship 1:

Customer → ShoppingCart: There is a composition relationship between Customer and ShoppingCart. Each Customer has a ShoppingCart that holds selected e-books. The ShoppingCart is exclusive to the Customer, meaning if the Customer is deleted, their ShoppingCart is also removed, representing strong ownership.

Relationship 2:

ShoppingCart → Ebook: There is an aggregation relationship between ShoppingCart and Ebook. A ShoppingCart can contain multiple Ebooks, but each Ebook exists independently outside the cart. This relationship shows that ShoppingCart groups e-books for purchase but does not "own" them in the system.

Relationship 3:

Order → Ebook: There is an aggregation relationship between Order and Ebook, indicating that an order can contain multiple e-books. Each Ebook is added to the order upon purchase, but it exists independently within the catalog. This shows that Order references e-books without creating a dependency on them.

Relationship 4:

Order → Discount: There is an aggregation relationship between Order and Discount, allowing multiple discounts to apply to an order. Discounts can exist outside the Order and be applied dynamically. This makes it easy to manage and customize discounts independently of the order.

Relationship 5:

Order → Payment: There is an association relationship between Order and Payment. The Order triggers a Payment process once finalized, linking the purchase details to the payment information. Payment references order details but does not create a dependency on them, allowing payments to be managed separately.

Relationship 6:

Order → VAT: There is an aggregation relationship between Order and VAT. The Order includes VAT as part of the final amount, but the VAT calculation exists independently. This relationship provides flexibility in applying VAT without embedding tax calculations directly in the order.

Relationship 7:

Customer → BankAccount: There is an aggregation relationship between Customer and BankAccount, allowing each customer to link multiple bank accounts. These accounts exist independently of the customer, meaning they can be managed separately if needed. This relationship supports flexibility in linking financial information to customers.

Relationship 8:

Discount → LoyaltyDiscount: There is an inheritance relationship between Discount and LoyaltyDiscount. LoyaltyDiscount is a specialized type of Discount that applies additional discounts for loyalty program members, extending the base discount functionality.

- ## Assumptions:

1. **One Shopping Cart per Customer**:
    - Each `Customer` has a single `ShoppingCart` that can hold multiple e-books. This `ShoppingCart` is unique to the customer, and it is assumed that if the customer is deleted, the associated `ShoppingCart` is also removed.

2. **Automatic Application of Discounts Based on Conditions**:
    - The `Order` class automatically applies discounts based on predefined conditions. A 10% loyalty discount is applied if the customer is a loyalty program member, and a 20% bulk discount is applied if the order contains 5 or more e-books. These discounts are managed independently and can both apply to a single order if both conditions are met.
3. **Standard VAT Rate Applied to All Orders**:
    - Every order includes a VAT calculated at a fixed rate of 8%, managed by the `VAT` class. This VAT rate is consistent across all purchases and cannot be adjusted per customer or order, ensuring uniformity in tax calculations.
4. **Instant Order Completion and Delivery**:
    - Once the `Payment` is processed, the `Order` is marked as complete, and the customer immediately gains access to the purchased e-books. There is no delay in delivery, as the system assumes instant access to digital products upon successful payment.
5. **Bank Account Information Linked to Customer**:
    - Each `Customer` can link one or more `BankAccount` objects containing account details, including account number, bank name, and balance. This information is used solely for record-keeping and is not involved in actual payment processing within the system.

- **Write Python Code to Implement Your UML Class Diagram Implement the Python classes based on your UML diagram. Make sure to:**
    - **Group the classes into different files for good modularity.**
    - **Use docstring, private/protected attributes, getter and setter methods, and a __str__() method for all classes.**
    - **Include comments and documentation to ensure good readability.**
    - **The submitted code must be error-free and have well-formatted output. Each student's work will be unique in terms of the code and output.**

ebook.py

```
class Ebook:

    """Represents an e-book in the e-bookstore."""

    def __init__(self, title, author, publication_date, genre, price):

        self.__title = title
```

```
    self.__author = author
    self.__publication_date = publication_date
    self.__genre = genre
    self.__price = price

def get_title(self):
    return self.__title

def set_title(self, title):
    self.__title = title

def get_author(self):
    return self.__author

def set_author(self, author):
    self.__author = author

def get_price(self):
```

```
        return self.__price



    def set_price(self, price):
        self.__price = price



    def __str__(self):
        return f"Ebook: {self.__title} by {self.__author} - {self.__genre} (${self.__price})"
```

customer.py

```
from shopping_cart import ShoppingCart
from order import Order
from bank_account import BankAccount


class Customer:
    """Represents a customer in the e-bookstore."""


    def __init__(self, name, contact_info):
```

```
        self.__name = name
        self.__contact_info = contact_info
        self._shopping_cart = ShoppingCart()
        self.__orders = []
        self.__accounts = []

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    def get_contact_info(self):
        return self.__contact_info

    def set_contact_info(self, contact_info):
        self.__contact_info = contact_info
```

```
    def add_order(self, order):
        self.__orders.append(order)

    def get_orders(self):
        return self.__orders

    def add_account(self, account):
        if isinstance(account, BankAccount):
            self.__accounts.append(account)

    def get_accounts(self):
        return self.__accounts

    def __str__(self):
        return f"Customer: {self.__name} ({self.__contact_info})"

customer = Customer("Ali Abdullah", "AA.B@example.com")
print(customer)  # This should show a formatted output if __str__() is implemented correctly
```

Shopping_cart.py

```
from ebook import Ebook



class ShoppingCart:

    """Represents a shopping cart for a customer."""



    def __init__(self):

        self.__ebooks = []



    def add_ebook(self, ebook):

        self.__ebooks.append(ebook)



    def remove_ebook(self, ebook):

        self.__ebooks.remove(ebook)



    def calculate_total(self):
```

```
        return sum(ebook.get_price() for ebook in self.__ebooks)


    def __str__(self):
        return "Shopping Cart: " + ", ".join([str(ebook) for ebook in self.__ebooks])
```

order.py

```
from ebook import Ebook
from vat import VAT
from discount import Discount
from loyalty_discount import LoyaltyDiscount


class Order:
    """Represents an order placed by a customer, with support for discounts and VAT."""


    def __init__(self, order_date, ebooks):
        self.__order_date = order_date
        self.__ebooks = ebooks
```

```
        self.__total_amount = sum(ebook.get_price() for ebook in ebooks)
        self.__discount = 0.0
        self.__vat = VAT()
        self.__loyalty_discount = None
        self.__bulk_discount_applied = False

        # Check for bulk discount eligibility
        if len(ebooks) >= 5:
            self.apply_bulk_discount()

    def get_order_date(self):
        return self.__order_date

    def set_discount(self, discount):
        """Applies a general discount to the order if provided."""
        if isinstance(discount, Discount):
            discount_amount = discount.calculate_discount(self.__total_amount)
            self.__discount += discount_amount
```

```
            self.__total_amount -= discount_amount
        else:
            print("Invalid discount type provided.")


    def apply_bulk_discount(self):
        """Applies a 20% bulk discount if 5 or more e-books are in the order."""
        bulk_discount = 0.20  # 20% bulk discount
        self.__discount += self.__total_amount * bulk_discount
        self.__total_amount -= self.__total_amount * bulk_discount
        self.__bulk_discount_applied = True


    def set_loyalty_discount(self, loyalty_discount):
        """Applies a loyalty discount if the customer qualifies."""
        if isinstance(loyalty_discount, LoyaltyDiscount):
            discount_amount = loyalty_discount.calculate_discount(self.__total_amount)
            self.__discount += discount_amount
            self.__total_amount -= discount_amount
            assert isinstance(loyalty_discount, object)
```

```
            self.__loyalty_discount = loyalty_discount



    def get_total_with_vat(self):

        """Returns the total amount including VAT."""

        return self.__vat.calculate_total_with_vat(self.__total_amount)



    def __str__(self):

        ebooks_str = ", ".join([ebook.get_title() for ebook in self.__ebooks])

        discount_info = "Bulk Discount Applied" if self.__bulk_discount_applied else "No Bulk
Discount"

        loyalty_info = f"Loyalty Discount: {self.__loyalty_discount}" if self.__loyalty_discount else "No
Loyalty Discount"

        return (f"Order Date: {self.__order_date}\nEbooks: [{ebooks_str}]\n"

                f"Subtotal after discounts: ${self.__total_amount}\n{discount_info}\n{loyalty_info}\n"

                f"Total with VAT: ${self.get_total_with_vat()}")
```

payment.py

```
from ebook import Ebook
```

```
class Payment:

    """Represents a payment for an order."""



    def __init__(self, order_date, ebooks, total_amount, discount=0.0):

        self.__order_date = order_date

        self.__ebooks = ebooks

        self.__total_amount = total_amount

        self.__discount = discount



    def process_payment(self):

        # Simulate payment processing

        self.__status = "Completed"



    def __str__(self):

        ebooks_str = ", ".join([ebook.get_title() for ebook in self.__ebooks])

        return f"Payment Date: {self.__order_date}, Ebooks: [{ebooks_str}], Total:
${self.__total_amount - self.__discount}"
```

discount.py

```
class Discount:

    """Represents a generic discount."""



    def __init__(self, discount_rate):

        self.__discount_rate = discount_rate



    def calculate_discount(self, total_amount):

        """Calculates the discount based on the given rate."""

        return total_amount * self.__discount_rate





    def __str__(self):

        return f"Discount: {self.__discount_rate * 100}%"
```

Loyalty_discount.py

```
from discount import Discount

class LoyaltyDiscount(Discount):

    """Represents a loyalty discount for frequent customers."""



    def __init__(self, discount_rate, loyalty_points):

        super().__init__(discount_rate)

        self.__loyalty_points = loyalty_points



    def get_loyalty_points(self):

        return self.__loyalty_points



    def __str__(self):

        return f"Loyalty Discount: {self._Discount__discount_rate * 100}% with
{self.__loyalty_points} points required"
```

bank_account.py

```
class BankAccount:
    """Represents a bank account associated with a customer."""

    def __init__(self, account_number, bank_name, balance):
        self.__account_number = account_number
        self.__bank_name = bank_name
        self.__balance = balance

    def get_account_number(self):
        return self.__account_number

    def set_account_number(self, account_number):
        self.__account_number = account_number

    def get_balance(self):
        return self.__balance

    def set_balance(self, balance):
```

```
        self.__balance = balance



    def __str__(self):

        return f"BankAccount: {self.__account_number} at {self.__bank_name} - Balance:
${self.__balance}"
```

vat.py

```
class VAT:

    """Represents VAT calculation for the e-bookstore."""

    def __init__(self, vat_rate=0.08):

        """Initialize with a default VAT rate of 8% (0.08)."""

        self.__vat_rate = vat_rate



    def get_vat_rate(self):

        """Returns the current VAT rate."""

        return self.__vat_rate



    def set_vat_rate(self, vat_rate):

        """Sets a new VAT rate."""
```

```
    self.__vat_rate = vat_rate



def calculate_vat(self, amount):

    """Calculates the VAT for a given amount."""

    return amount * self.__vat_rate



def calculate_total_with_vat(self, amount):

    """Calculates the total amount including VAT."""

    return amount + self.calculate_vat(amount)



def __str__(self):

    return f"VAT Rate: {self.__vat_rate * 100}%"
```

- **Define Test Cases Write test cases in a separate file to demonstrate all the program's features. Ensure to test <u>all features</u>. Some examples of test cases are (not limited to):**
    - **Add/Modify/Remove a new e-book to the e-bookstore's catalog.**

```
def test_ebook_catalog():
    """Test adding, modifying, and removing an e-book in the catalog."""
    print("\n=== Test: E-book Catalog ===")
    ebook1 = Ebook("Python Basics", "Alice Smith", "2022-01-10", "Programming", 25.99)
    ebook2 = Ebook("Data Science Essentials", "John Doe", "2021-08-20", "Data Science", 35.50)



    # Modify the e-book
    ebook1.set_price(20.99)
    print(ebook1)  # Output modified e-book details



    # Add new e-book to the catalog
    print(ebook2)  # Display second e-book
```

    - **Add/Modify/Remove customer account.**

```
def test_customer_account():
```

```
    """Test adding, modifying, and viewing customer accounts."""
    print("\n=== Test: Customer Account ===")
    customer = Customer("Bob Johnson", "bob@example.com")

    # Modify customer contact info
    customer.set_contact_info("bob.johnson@newemail.com")
    print(customer)

    # Add bank account to customer
    account = BankAccount("1234567890", "Bank of Python", 1000.0)
    customer.add_account(account)
    print("Bank Accounts:", [str(acc) for acc in customer.get_accounts()])
```

- **The addition of e-books to the shopping cart.**

```
def test_shopping_cart():
    """Test adding and removing e-books in the shopping cart."""
    print("\n=== Test: Shopping Cart ===")
    customer = Customer("Eve Adams", "eve@example.com")
```

```
    ebook1 = Ebook("Python Basics", "Alice Smith", "2022-01-10", "Programming", 25.99)

    ebook2 = Ebook("Data Science Essentials", "John Doe", "2021-08-20", "Data Science", 35.50)



    # Add e-books to shopping cart

    customer._shopping_cart.add_ebook(ebook1)

    customer._shopping_cart.add_ebook(ebook2)

    print(customer._shopping_cart)



    # Remove an e-book from the shopping cart

    customer._shopping_cart.remove_ebook(ebook1)

    print("After removing an e-book:", customer._shopping_cart)
```

- **Applying discounts for loyalty program members or bulk purchases.**

```
def test_apply_discounts():

    """Test applying a loyalty discount and a bulk discount for eligible orders."""

    print("\n=== Test: Applying Discounts ===")
```

```
    ebook1 = Ebook("Python Basics", "Mohammad Abdullah", "2022-01-10",
"Programming", 25.99)

    ebook2 = Ebook("Advanced Python", "Ahmed Waleed", "2023-05-15", "Programming",
29.99)

    ebook3 = Ebook("Data Science Essentials", "Alia Ibrahim", "2021-08-20", "Data
Science", 35.50)

    ebook4 = Ebook("AI Basics", "Maryam Saif", "2020-03-12", "Artificial Intelligence",
40.00)

    ebook5 = Ebook("Machine Learning", "Sujith Mathew", "2021-11-22", "Machine
Learning", 45.00)

    ebooks = [ebook1, ebook2, ebook3, ebook4, ebook5]




    order = Order("2024-11-11", ebooks)

    print("Order with Bulk Discount Applied:")

    print(order)  # This should show the bulk discount (20%) applied automatically




    loyalty_discount = LoyaltyDiscount(0.10, 100)  # 10% loyalty discount for members
with 100 points
```

```
    order.set_loyalty_discount(loyalty_discount)

    print("\nOrder with Bulk and Loyalty Discount Applied:")

    print(order)  # This should show both discounts applied to the final total
```

- **The generation of an invoice showing relevant discounts and required payments.**

```
def test_payment_invoice():

    """Test generating an invoice with payment processing, including discounts and VAT."""

    print("\n=== Test: Payment and Invoice Generation ===")



    # Create five e-books to trigger bulk discount

    ebook1 = Ebook("Python Basics", "Mohammad Abdullah", "2022-01-10", "Programming", 25.99)

    ebook2 = Ebook("Advanced Python", "Ahmed Waleed", "2023-05-15", "Programming", 29.99)

    ebook3 = Ebook("Data Science Essentials", "Alia Ibrahim", "2021-08-20", "Data Science", 35.50)

    ebook4 = Ebook("AI Basics", "Maryam Saif", "2020-03-12", "Artificial Intelligence", 40.00)

    ebook5 = Ebook("Machine Learning", "Sujith Mathew", "2021-11-22", "Machine Learning", 45.00)
```

```
ebooks = [ebook1, ebook2, ebook3, ebook4, ebook5]



# Create order and apply bulk discount automatically

order = Order("2024-11-11", ebooks)

discount = Discount(0.20)  # Apply an additional 20% general discount

order.set_discount(discount)  # This should now work without errors

print("Order with Bulk and Additional Discount Applied:", order)



# Process payment with VAT included

vat = VAT(0.08)

total_with_vat = vat.calculate_total_with_vat(order.get_total_with_vat())

payment = Payment(order.get_order_date(), ebooks, total_with_vat,

                  discount.calculate_discount(order.get_total_with_vat()))

payment.process_payment()

print("Payment Invoice:", payment)  # Should show total with discounts and VAT
```

FULL Test Case Code

```
from ebook import Ebook
```

```
from customer import Customer

from shopping_cart import ShoppingCart

from order import Order

from payment import Payment

from discount import Discount

from loyalty_discount import LoyaltyDiscount

from bank_account import BankAccount

from vat import VAT



def test_ebook_catalog():

    """Test adding, modifying, and removing an e-book in the catalog."""

    print("\n=== Test: E-book Catalog ===")

    ebook1 = Ebook("Python Basics", "Alice Smith", "2022-01-10", "Programming", 25.99)

    ebook2 = Ebook("Data Science Essentials", "John Doe", "2021-08-20", "Data Science", 35.50)



    # Modify the e-book

    ebook1.set_price(20.99)

    print(ebook1)  # Output modified e-book details
```

```
    # Add new e-book to the catalog
    print(ebook2)  # Display second e-book


def test_customer_account():
    """Test adding, modifying, and viewing customer accounts."""
    print("\n=== Test: Customer Account ===")
    customer = Customer("Bob Johnson", "bob@example.com")

    # Modify customer contact info
    customer.set_contact_info("bob.johnson@newemail.com")
    print(customer)

    # Add bank account to customer
    account = BankAccount("1234567890", "Bank of Python", 1000.0)
    customer.add_account(account)
    print("Bank Accounts:", [str(acc) for acc in customer.get_accounts()])
```

```
def test_shopping_cart():
    """Test adding and removing e-books in the shopping cart."""
    print("\n=== Test: Shopping Cart ===")
    customer = Customer("Eve Adams", "eve@example.com")
    ebook1 = Ebook("Python Basics", "Alice Smith", "2022-01-10", "Programming", 25.99)
    ebook2 = Ebook("Data Science Essentials", "John Doe", "2021-08-20", "Data Science", 35.50)

    # Add e-books to shopping cart
    customer._shopping_cart.add_ebook(ebook1)
    customer._shopping_cart.add_ebook(ebook2)
    print(customer._shopping_cart)

    # Remove an e-book from the shopping cart
    customer._shopping_cart.remove_ebook(ebook1)
    print("After removing an e-book:", customer._shopping_cart)


def test_apply_discounts():
```

```
"""Test applying a loyalty discount and a bulk discount for eligible orders."""

print("\n=== Test: Applying Discounts ===")



ebook1 = Ebook("Python Basics", "Mohammad Abdullah", "2022-01-10", "Programming", 25.99)

ebook2 = Ebook("Advanced Python", "Ahmed Waleed", "2023-05-15", "Programming", 29.99)

ebook3 = Ebook("Data Science Essentials", "Alia Ibrahim", "2021-08-20", "Data Science", 35.50)

ebook4 = Ebook("AI Basics", "Maryam Saif", "2020-03-12", "Artificial Intelligence", 40.00)

ebook5 = Ebook("Machine Learning", "Sujith Mathew", "2021-11-22", "Machine Learning", 45.00)

ebooks = [ebook1, ebook2, ebook3, ebook4, ebook5]



order = Order("2024-11-11", ebooks)

print("Order with Bulk Discount Applied:")

print(order)  # This should show the bulk discount (20%) applied automatically



loyalty_discount = LoyaltyDiscount(0.10, 100)  # 10% loyalty discount for members with 100 points
```

```
    order.set_loyalty_discount(loyalty_discount)
    print("\nOrder with Bulk and Loyalty Discount Applied:")
    print(order)  # This should show both discounts applied to the final total


def test_payment_invoice():
    """Test generating an invoice with payment processing, including discounts and VAT."""
    print("\n=== Test: Payment and Invoice Generation ===")

    # Create five e-books to trigger bulk discount
    ebook1 = Ebook("Python Basics", "Mohammad Abdullah", "2022-01-10", "Programming", 25.99)
    ebook2 = Ebook("Advanced Python", "Ahmed Waleed", "2023-05-15", "Programming", 29.99)
    ebook3 = Ebook("Data Science Essentials", "Alia Ibrahim", "2021-08-20", "Data Science", 35.50)
    ebook4 = Ebook("AI Basics", "Maryam Saif", "2020-03-12", "Artificial Intelligence", 40.00)
    ebook5 = Ebook("Machine Learning", "Sujith Mathew", "2021-11-22", "Machine Learning", 45.00)
    ebooks = [ebook1, ebook2, ebook3, ebook4, ebook5]

    # Create order and apply bulk discount automatically
```

```
    order = Order("2024-11-11", ebooks)
    discount = Discount(0.20)  # Apply an additional 20% general discount
    order.set_discount(discount)  # This should now work without errors
    print("Order with Bulk and Additional Discount Applied:", order)

    # Process payment with VAT included
    vat = VAT(0.08)
    total_with_vat = vat.calculate_total_with_vat(order.get_total_with_vat())
    payment = Payment(order.get_order_date(), ebooks, total_with_vat,
                      discount.calculate_discount(order.get_total_with_vat()))
    payment.process_payment()
    print("Payment Invoice:", payment)  # Should show total with discounts and VAT


if __name__ == "__main__":
    # Run all test cases
    test_ebook_catalog()
    test_customer_account()
```

test_shopping_cart()

test_apply_discounts()

test_payment_invoice()

- **Summary:**

In this assignment, I worked on creating different classes with their own methods and attributes. I used a UML diagram to show how each class is connected, using numbers or “*” symbols to show the quantity of each relationship. For each class, I wrote Python code with five attributes and made objects and instances to show how each class works in the system. I also made test cases to show how the system can be changed and adjusted. This helped me understand the different relationships between classes. Finally, I shared my work on GitHub to show my progress and the working Python code.

GiThub Link:

https://github.com/MohammadAbdullahHashim/220ASS2.git