Assignment 2

Professor Sujith Mathew

ICS220 programming fundamentals

Zayed University

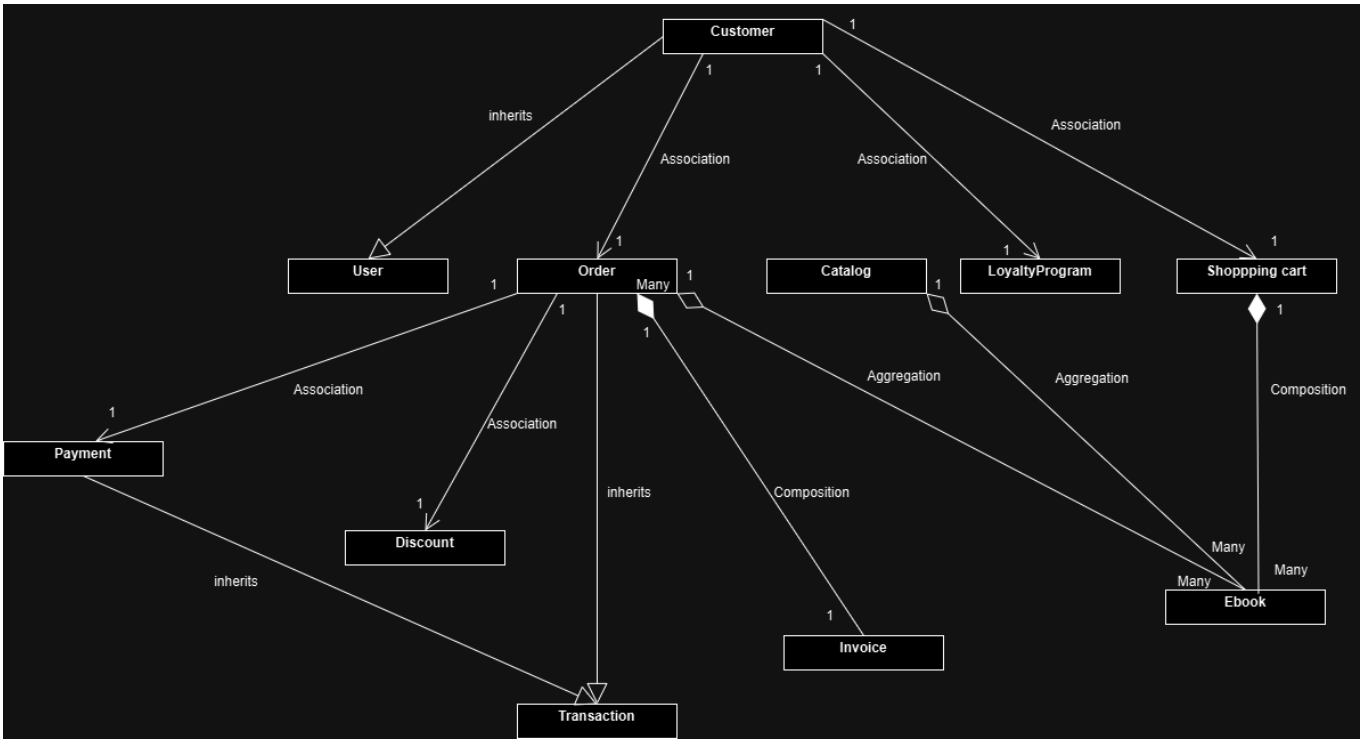
Al Hamzah Zohdi Nassar

11/4/2024

E-book Store Management system:

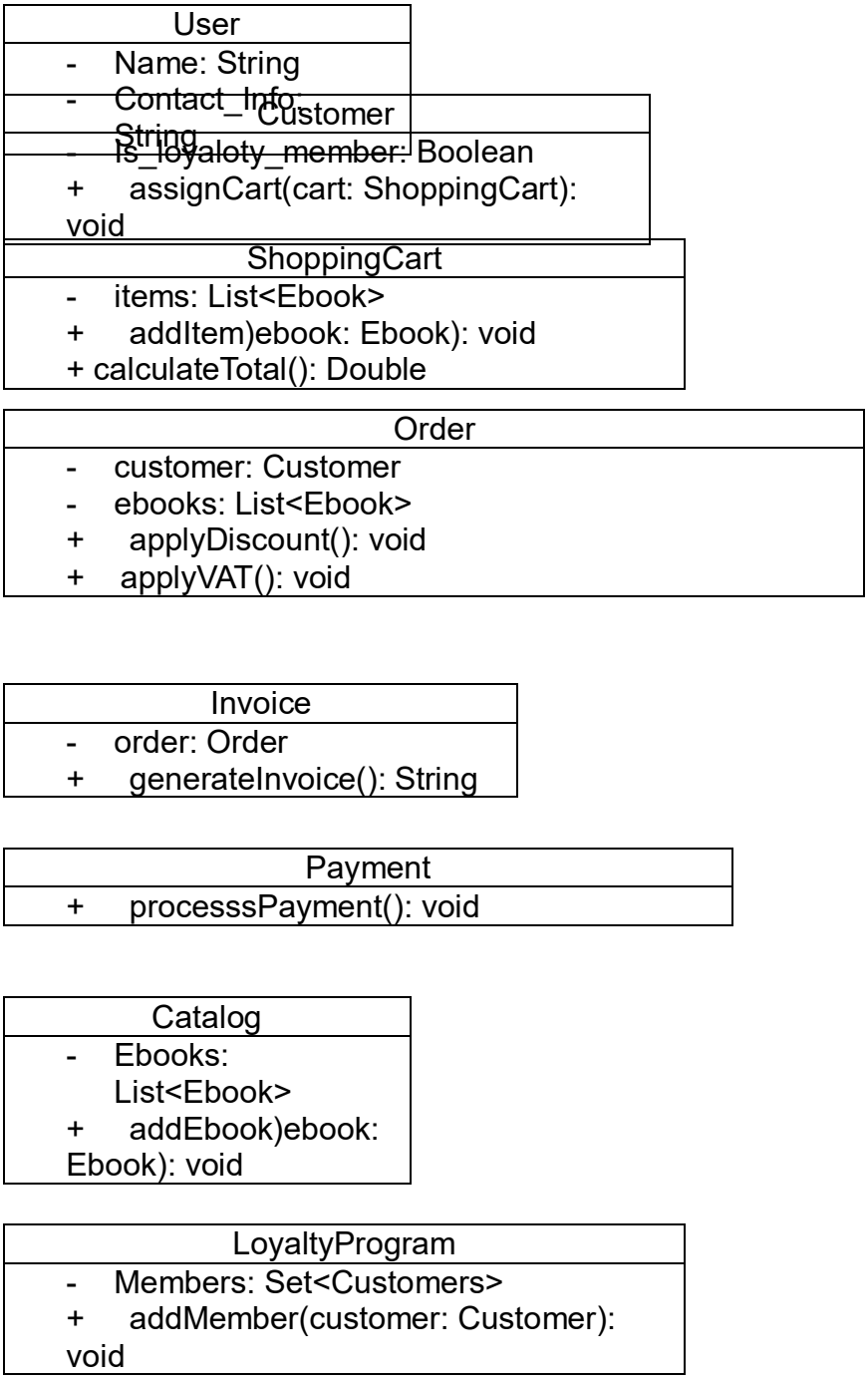
Part 1

UML Class Diagrams:



Part 2

Class Diagram (Attributes / Methods):



Ebook
- title: String - author: String - publication_date: String - genre: String - price: Double + getPrice(): Double

Transaction
- amount: Double - status: String + process(): void

Discount
- percentage: Double + apply(price: Double): Double

Part 3

UML class Description

User:

- Represents a generic user of the system, containing basic information such as name and contact_info.
- Acts as a base class for **Customer**, meaning every customer "is a" user with basic attributes.

Customer (inherits from User):

- Represents a specific type of User who can interact with the e-bookstore by adding items to their cart, placing orders, and participating in the loyalty program.
- Has a ShoppingCart for holding selected e-books and is "associated with" multiple orders.
- "Is a" type of User with extended attributes, such as is_loyalty_member, to determine if the customer is eligible for discounts.

Ebook:

- Represents an individual e-book, with attributes like title, author, publication_date, genre, and price.
- A ShoppingCart "has multiple" e-books, and each Order also "contains multiple" e-books.
- Aggregated in the Catalog as part of the store's available collection.

Catalog:

- Represents the collection of all e-books available in the store.
- Aggregates multiple Ebook entries, meaning a catalog "has multiple" e-books.
- Provides a method to add e-books to the collection.

ShoppingCart:

- Temporarily holds the selected Ebook items that a Customer wants to purchase.
- Each ShoppingCart "is associated with" a single Customer, and it "composes" multiple e-books (the e-books are part of the cart and exist as items within it).
- Provides methods for adding, removing, and calculating the total of the e-books.

Transaction:

- Represents a base class for financial operations in the system, containing general transaction attributes such as amount and status.
- Acts as a parent class for both Order and Payment, so each order and payment "is a" transaction and inherits these shared attributes and methods.

Order (inherits from Transaction):

- Represents a finalized purchase made by a Customer.
- Contains details about the items being ordered and the customer associated with the order.
- "Is a" type of Transaction with additional order-specific methods such as applyDiscount and applyVAT.
- Aggregates Ebook items from the ShoppingCart (an order "includes multiple" e-books).
- Each order "is associated with" an Invoice (which composes the order details) and may apply a Discount.

Invoice:

- Generates an itemized summary for each Order.
- "Is composed of" a single Order, meaning it is tightly coupled with one specific order to generate its details.
- Provides a method to generate an invoice based on the order's final details.

Discount:

- Represents discount options that can be applied to an Order.
- "Is associated with" an order, allowing for either a loyalty discount or bulk purchase discount to be calculated based on specific criteria.
- Can calculate a discount based on a percentage applied to the order’s total.

Payment (inherits from Transaction):

- Represents the payment process for an order, with attributes like amount and status, inherited from Transaction.
- "Is a" type of Transaction with specific functionality to processPayment.
- "Is associated with" a single Order, as each payment is tied to one specific order.

LoyaltyProgram:

- Manages loyalty membership for customers in the e-bookstore.
- "Is associated with" multiple customers, meaning customers can be part of the loyalty program.
- Allows for the addition of members and tracks loyalty status, which can influence discount eligibility.

Part 4

Code:

Classes:

```
# ebook store.py

class Ebook:
    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 getPrice(self):
        return self.price

    def __str__(self):
        return f"Ebook(title={self.title}, author={self.author}, price={self.price})"


class User:
    def __init__(self, name, contact_info):
        self.name = name
        self.contact_info = contact_info


class Customer(User):
    def   init  (self, name, contact info, is loyalty member=False):
        super().__init__(name, contact_info)
        self.is_loyalty_member = is_loyalty_member

    def __str__(self):
        return f"Customer(name={self.name}, contact info={self.contact info},
loyalty_member={self.is_loyalty_member})"


class ShoppingCart:
    def   init  (self):
        self.items = []

    def addItem(self, ebook):
        self.items.append(ebook)
        print(f"Added {ebook.title} to cart.")

    def removeItem(self, ebook):
        if ebook in self.items:
            self.items.remove(ebook)
            print(f"Removed {ebook.title} from cart.")
        else:
            print(f"{ebook.title} not in cart.")

    def calculateTotal(self):
        return sum(item.getPrice() for item in self.items)

    def   str  (self):
        return f"ShoppingCart(items={[str(item) for item in self.items]})"


class Transaction:
    def   init  (self, amount):
        self.amount = amount
        self.status = "Pending"

    def process(self):
        if self.amount > 0:
            self.status = "Completed"
        else:
            self.status = "Failed"


class Order(Transaction):
    VAT_RATE = 0.08
    LOYALTY_DISCOUNT = 0.10
    BULK_DISCOUNT = 0.20

    def   init  (self, customer, ebooks):
        self.customer = customer
        self.ebooks = ebooks
        amount = sum(book.getPrice() for book in ebooks)
        super().__init__(amount)
```

```
    def applyDiscount(self):
        if len(self.ebooks) >= 5:
            self.amount *= (1 - self.BULK DISCOUNT)
        elif self.customer.is loyalty member:
            self.amount *= (1 - self.LOYALTY_DISCOUNT)

    def applyVAT(self):
        self.amount *= (1 + self.VAT RATE)

    def __str__(self):
        return f"Order(customer={self.customer}, amount={self.amount}, status={self.status})"


class Invoice:
    def   init  (self, order):
        self.order = order

    def generateInvoice(self):
        items = "\n".join([f"{ebook.title} - ${ebook.getPrice():.2f}" for ebook in
self.order.ebooks])
        total line = f"Total after discounts and VAT: ${self.order.amount:.2f}"
        return f"Invoice for {self.order.customer.name}:\n{items}\n{total_line}"

    def __str__(self):
        return self.generateInvoice()


class Catalog:
    def __init__(self):
        self.ebooks = []

    def addEbook(self, ebook):
        self.ebooks.append(ebook)
        print(f"Added {ebook.title} to the catalog.")

    def removeEbook(self, ebook):
        if ebook in self.ebooks:
            self.ebooks.remove(ebook)
            print(f"Removed {ebook.title} from the catalog.")
        else:
            print(f"{ebook.title} not found in the catalog.")

    def __str__(self):
        return f"Catalog(ebooks={[str(ebook) for ebook in self.ebooks]})"


class LoyaltyProgram:
    def __init__(self):
        self.members = set()

    def addMember(self, customer):
        self.members.add(customer)
        customer.is_loyalty_member = True
        print(f"{customer.name} has been added to the loyalty program.")
```

test_catalog.py:

```
# test_catalog.py
from ebook_store import Catalog, Ebook

def test_catalog_management():
    # Initialize catalog
    catalog = Catalog()

    # Add a new e-book to the catalog
    ebook1 = Ebook("Harry Potter", "JK Rowling", "1999-01-12", "Fantasy",
```

```
29.99)
    ebook2 = Ebook("Lord Of The Rings", "John Ronald Reuel Tolkien", "1973-
02-9", "Fantasy", 39.99)
    catalog.addEbook(ebook1)
    catalog.addEbook(ebook2)

    # Show catalog contents
    print("\nCatalog contents after adding two e-books:")
    print(catalog)

    # Modify the price of an e-book in the catalog
    print("\nModifying price of the first e-book...")
    ebook1.price = 24.99
    print(f"Updated {ebook1.title} price to ${ebook1.price:.2f}")

    # Show catalog contents after modification
    print("\nCatalog contents after modifying e-book price:")
    print(catalog)

    # Remove an e-book from the catalog
    print("\nRemoving an e-book from the catalog...")
    catalog.removeEbook(ebook1)

    # Show catalog contents after removal
    print("\nCatalog contents after removing an e-book:")
    print(catalog)

# Run the test
if __name__ == "__main__":
    test_catalog_management()
```

test_cutomer_management.py:

```
# test_customer_management.py
from ebook_store import Customer, LoyaltyProgram

def test_customer_management():
    # Initialize a loyalty program
    loyalty_program = LoyaltyProgram()

    # Add a new customer
    customer1 = Customer("Al Hamzah Nassar", "alhamzahnassar@gmail.com",
is_loyalty_member=False)
```

```
    customer2 = Customer("Adnan Daher", "adnandaher@gmail.com",
is_loyalty_member=False)
    print("\nAdding new customers:")
    print(customer1)
    print(customer2)

    # Add customers to the loyalty program
    print("\nAdding Al Hamzah Nassar to the loyalty program...")
    loyalty_program.addMember(customer1)

    # Show updated customer information
    print("\nCustomer information after adding to loyalty program:")
    print(customer1)
    print(customer2)

    # Modify customer information
    print("\nModifying Adnan Daher's contact information...")
    customer2.contact_info = "adnandaher01@gmail.com"
    print("Updated customer information for Adnan Daher:")
    print(customer2)

    # Simulate removing a customer (simply by setting to None or ignoring
further use in this test)
    print("\nRemoving customer Al Hamzah Nassar from the system...")
    del customer1  # Note: In a real system, we would remove references and
clear data.

    print("\nRemaining customer information in the system:")
    print(customer2)

# Run the test
if __name__ == "__main__":
    test_customer_management()
```

test_discount_application.py:

```
# test_discount_application.py
from ebook_store import Customer, Ebook, ShoppingCart, Order


def test_discount_application():
    # Create e-books to add to the cart
    ebook1 = Ebook("Harry Potter", "JK Rowling", "1999-01-12", "Fantasy",
29.99)
    ebook2 = Ebook("Lord Of The Rings", "John Ronald Reuel Tolkien", "1987-
02-09", "Fantasy", 39.99)
    ebook3 = Ebook("Rich Dad Poor Dad", "Robert Kiyosaki", "1997-02-04",
```

```
"Philosophy", 49.99)
    ebook4 = Ebook("Diary Of A Wimpy Kid", "Jeff Kinney", "2000-01-06",
"Children", 59.99)
    ebook5 = Ebook("Matilda", "Roald Dahl.", "1987-08-11", "Children", 69.99)

    # Initialize a shopping cart and add e-books
    cart = ShoppingCart()
    cart.addItem(ebook1)
    cart.addItem(ebook2)
    cart.addItem(ebook3)
    cart.addItem(ebook4)
    cart.addItem(ebook5)

    # Show cart contents and total before applying discounts
    print("\nShopping cart contents before applying discounts:")
    print(cart)
    print(f"Total price: ${cart.calculateTotal():.2f}")

    # Create a customer who is a loyalty program member
    customer = Customer("Alice", "alice@example.com", is_loyalty_member=True)

    # Create an order with the shopping cart items for a loyalty program
member
    print("\nCreating an order for a loyalty program member...")
    order = Order(customer, cart.items)
    order.applyDiscount()  # Applies a 10% loyalty discount for Alice
    print(f"Total after loyalty discount: ${order.amount:.2f}")

    # Show total with loyalty discount applied
    print("\nCreating an order for a bulk purchase (5 or more items)...")
    bulk_order = Order(customer, cart.items)
    bulk_order.applyDiscount()  # Applies a 20% bulk discount
    print(f"Total after bulk discount: ${bulk_order.amount:.2f}")


# Run the test
if __name__ == "__main__":
    test_discount_application()
```

test_invoice_generation.py:

```
# test_invoice_generation.py
from ebook_store import Customer, Ebook, ShoppingCart, Order, Invoice

def test_invoice_generation():
    # Create e-books to add to the cart
    ebook1 = Ebook("Harry Potter", "JK Rowling", "1999-01-12", "Fantasy",
29.99)
    ebook2 = Ebook("Lord Of The Rings", "John Ronald Reuel Tolkien", "1987-
02-09", "Fantasy", 39.99)
    ebook3 = Ebook("Rich Dad Poor Dad", "Robert Kiyosaki", "1997-02-04",
```

```
"Philosophy", 49.99)
    ebook4 = Ebook("Diary Of A Wimpy Kid", "Jeff Kinney", "2000-01-06",
"Children", 59.99)
    ebook5 = Ebook("Matilda", "Roald Dahl.", "1987-08-11", "Children", 69.99)

    # Initialize a shopping cart and add e-books
    cart = ShoppingCart()
    cart.addItem(ebook1)
    cart.addItem(ebook2)
    cart.addItem(ebook3)
    cart.addItem(ebook4)
    cart.addItem(ebook5)

    # Create a customer who is a loyalty program member
    customer = Customer("Alice", "alice@example.com", is_loyalty_member=True)

    # Create an order with the shopping cart items
    order = Order(customer, cart.items)
    print("\nOrder total before discounts and VAT:")
    print(f"${order.amount:.2f}")

    # Apply discounts and VAT
    order.applyDiscount()
    order.applyVAT()
    print("\nOrder total after discounts and VAT:")
    print(f"${order.amount:.2f}")

    # Generate and display invoice
    invoice = Invoice(order)
    print("\n--- Invoice ---")
    print(invoice.generateInvoice())

# Run the test
if __name__ == "__main__":
    test_invoice_generation()
```

test_shopping_cart.py:

```
# test_shopping_cart.py
from ebook_store import ShoppingCart, Ebook

def test_shopping_cart_management():
    # Initialize shopping cart
    cart = ShoppingCart()

    # Create e-books to add to the cart
    ebook1 = Ebook("Harry Potter", "JK Rowling", "1999-01-12", "Fantasy",
29.99)
    ebook2 = Ebook("Lord Of The Rings", "John Ronald Reuel Tolkien", "1987-
```

```
02-09", "Fantasy", 39.99)
    ebook3 = Ebook("Rich Dad Poor Dad", "Robert Kiyosaki", "1997-02-04",
"Philosophy", 49.99)

    # Add e-books to the shopping cart
    print("\nAdding e-books to the shopping cart:")
    cart.addItem(ebook1)
    cart.addItem(ebook2)
    cart.addItem(ebook3)

    # Show cart contents and total after adding items
    print("\nShopping cart contents after adding e-books:")
    print(cart)
    print(f"Total price: ${cart.calculateTotal():.2f}")

    # Remove an e-book from the shopping cart
    print("\nRemoving an e-book from the shopping cart:")
    cart.removeItem(ebook2)

    # Show cart contents and total after removal
    print("\nShopping cart contents after removing an e-book:")
    print(cart)
    print(f"Total price: ${cart.calculateTotal():.2f}")

# Run the test
if __name__ == "__main__":
    test_shopping_cart_management()
```

GitHub:

https://github.com/alhamzahnassar1/-Program.-Fund.-Assignment-2