Assignment 2

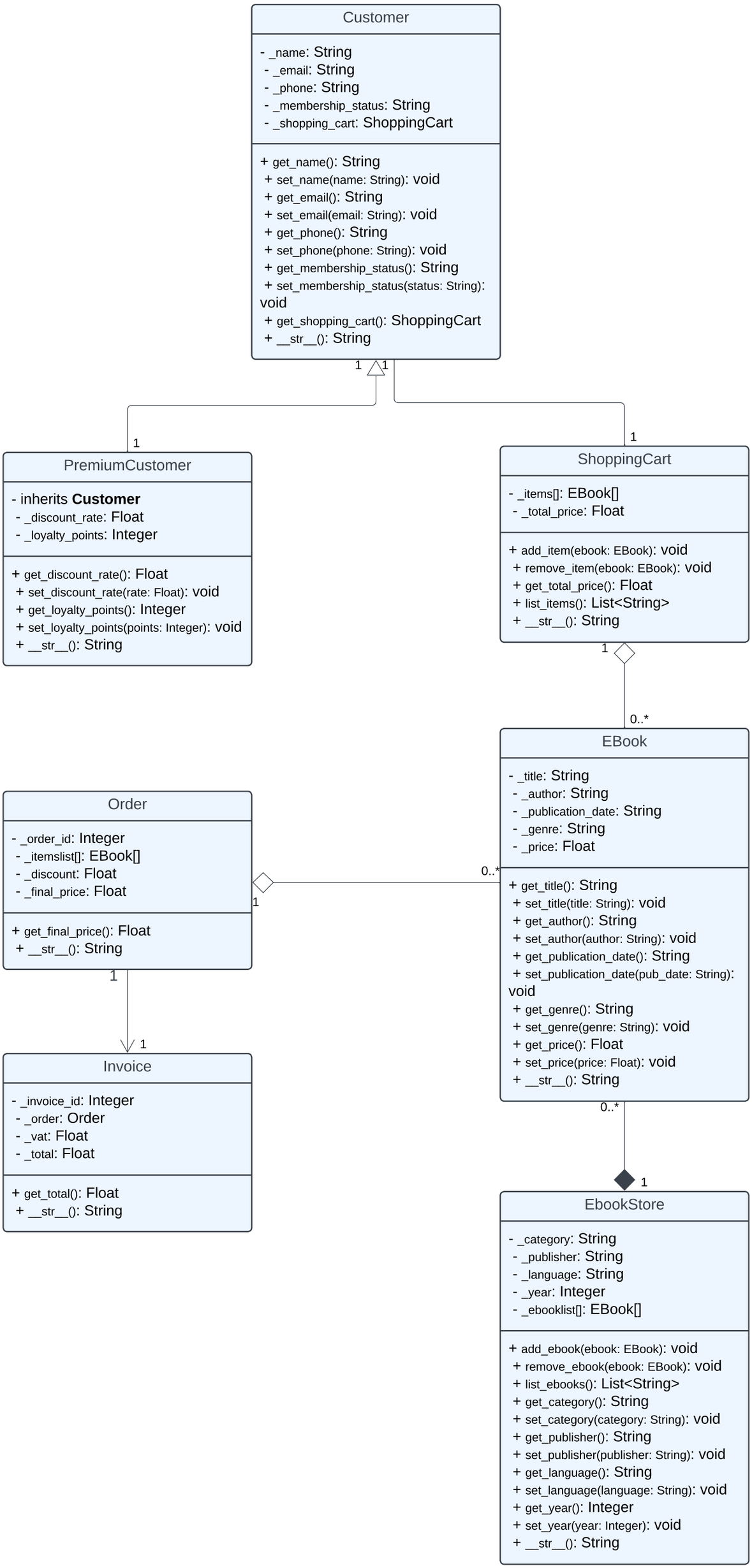
Saif A. Alhammadi: 202220746

College of Interdisciplinary Sciences

ICS220: Programming Fundamentals

Professor Parkar

04/11/2024


Customer
- _name: String
- _email: String
- _phone: String
- _membership_status: String
- _shopping_cart: ShoppingCart
+ get_name(): String
+ set_name(name: String): void
+ get_email(): String
+ set_email(email: String): void
+ get_phone(): String
+ set_phone(phone: String): void
+ get_membership_status(): String
+ set_membership_status(status: String): void
+ get_shopping_cart(): ShoppingCart
+ __str__(): String
1
1
1
1
PremiumCustomer
- inherits Customer
- _discount_rate: Float
- _loyalty_points: Integer
+ get_discount_rate(): Float
+ set_discount_rate(rate: Float): void
+ get_loyalty_points(): Integer
+ set_loyalty_points(points: Integer): void
+ __str__(): String
ShoppingCart
- _items[]: EBook[]
- _total_price: Float
+ add_item(ebook: EBook): void
+ remove_item(ebook: EBook): void
+ get_total_price(): Float
+ list_items(): List<String>
+ __str__(): String
1
0..*
EBook
- _title: String
- _author: String
- _publication_date: String
- _genre: String
- _price: Float
+ get_title(): String
+ set_title(title: String): void
+ get_author(): String
+ set_author(author: String): void
+ get_publication_date(): String
+ set_publication_date(pub_date: String): void
+ get_genre(): String
+ set_genre(genre: String): void
+ get_price(): Float
+ set_price(price: Float): void
+ __str__(): String
0..*
1
Order
- _order_id: Integer
- _itemslist[]: EBook[]
- _discount: Float
- _final_price: Float
+ get_final_price(): Float
+ __str__(): String
1
1
Invoice
- _invoice_id: Integer
- _order: Order
- _vat: Float
- _total: Float
+ get_total(): Float
+ __str__(): String
0..*
1
EbookStore
- _category: String
- _publisher: String
- _language: String
- _year: Integer
- _ebooklist[]: EBook[]
+ add_ebook(ebook: EBook): void
+ remove_ebook(ebook: EBook): void
+ list_ebooks(): List<String>
+ get_category(): String
+ set_category(category: String): void
+ get_publisher(): String
+ set_publisher(publisher: String): void
+ get_language(): String
+ set_language(language: String): void
+ get_year(): Integer
+ set_year(year: Integer): void
+ __str__(): String

Class 1	Class 2	Association Type	Cardinality	Relationship Description
Customer	ShoppingCart	Unary	1 (Customer) : 1 (ShoppingCart)	Each Customer has one associated ShoppingCart they manage directly.
Customer	PremiumCustomer	Inheritance	1 (Customer) : 1 (PremiumCustomer)	PremiumCustomer inherits from Customer, adding premium-specific attributes.
ShoppingCart	EBook	Aggregation	1 (ShoppingCart) : * (EBook)	A ShoppingCart can contain multiple EBook objects, which exist independently of the cart.
Order	EBook	Composition	1 (Order) : * (EBook)	Each Order is composed of multiple EBook items, representing purchased items that are integral to the order.
Order	Invoice	Binary	1 (Order) : 1 (Invoice)	Each Order is associated with a unique Invoice, detailing order charges and totals.
EbookStore	EBook	Aggregation	1 (EbookStore) : * (EBook)	Each EbookStore aggregates multiple Ebook objects, which represent the store’s catalog items.

Classes.py

```
# classes.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


    # Getters and Setters

    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_publication_date(self):

        return self._publication_date

    def set_publication_date(self, publication_date):

        self._publication_date = publication_date


    def get_genre(self):

        return self._genre

    def set_genre(self, genre):

        self._genre = genre


    def get_price(self):

        return self._price

    def set_price(self, price):

        self._price = price
```

```
    def __str__(self):
        return f"EBook({self._title}, {self._author},
{self._publication_date}, {self._genre}, ${self._price})"



class ShoppingCart:
    def __init__(self):
        self._items = []
        self._total_price = 0.0

    def add_item(self, ebook):
        self._items.append(ebook)
        self._total_price += ebook.get_price()

    def remove_item(self, ebook):
        if ebook in self._items:
            self._items.remove(ebook)
            self._total_price -= ebook.get_price()
```

```
    def get_total_price(self):
        return self._total_price

    def list_items(self):
        return [str(item) for item in self._items]

    def __str__(self):
        items_str = ', '.join([item.get_title() for item in self._items])
        return f"ShoppingCart(Items: [{items_str}], Total Price: ${self._total_price:.2f})"


class Customer:
    def __init__(self, name, email, phone, membership_status):
        self._name = name
        self._email = email
        self._phone = phone
        self._membership_status = membership_status
```

```
        self._shopping_cart = ShoppingCart()

    # Getters and Setters
    def get_name(self):
        return self._name
    def set_name(self, name):
        self._name = name

    def get_email(self):
        return self._email
    def set_email(self, email):
        self._email = email

    def get_phone(self):
        return self._phone
    def set_phone(self, phone):
        self._phone = phone

    def get_membership_status(self):
```

```
        return self._membership_status

    def set_membership_status(self, membership_status):

        self._membership_status = membership_status



    def get_shopping_cart(self):

        return self._shopping_cart



    def __str__(self):

        return f"Customer({self._name}, {self._email},
{self._membership_status})"




class PremiumCustomer(Customer):

    def __init__(self, name, email, phone, membership_status,
discount_rate, loyalty_points):

        super().__init__(name, email, phone,
membership_status)

        self._discount_rate = discount_rate

        self._loyalty_points = loyalty_points
```

```
    # Additional getters and setters
    def get_discount_rate(self):
        return self._discount_rate
    def set_discount_rate(self, discount_rate):
        self._discount_rate = discount_rate

    def get_loyalty_points(self):
        return self._loyalty_points
    def set_loyalty_points(self, loyalty_points):
        self._loyalty_points = loyalty_points

    def __str__(self):
        return f"PremiumCustomer({self._name}, {self._email},
{self._membership_status}, Discount Rate:
{self._discount_rate}, Loyalty Points:
{self._loyalty_points})"


class Order:
    def __init__(self, order_id, items, discount=0.0):
```

```
        self._order_id = order_id
        self._items = items
        self._discount = discount
        self._final_price = sum(item.get_price() for item in
items) * (1 - discount)

    def get_final_price(self):
        return self._final_price

    def __str__(self):
        items_str = ', '.join([item.get_title() for item in
self._items])
        return f"Order(ID: {self._order_id}, Items:
[{items_str}], Final Price: ${self._final_price:.2f})"


class Invoice:
    def __init__(self, invoice_id, order, vat_rate=0.08):
        self._invoice_id = invoice_id
        self._order = order
```

```
        self._vat = self._order.get_final_price() * vat_rate
        self._total = self._order.get_final_price() +
self._vat

    def get_total(self):
        return self._total

    def __str__(self):
        return f"Invoice(ID: {self._invoice_id}, Total (with
VAT): ${self._total:.2f})"


class EbookStore:
    def __init__(self, category, publisher, language, year):
        self._category = category
        self._publisher = publisher
        self._language = language
        self._year = year
        self._e_books = []
```

```
    # Getters and Setters

    def get_category(self):

        return self._category

    def set_category(self, category):

        self._category = category


    def get_publisher(self):

        return self._publisher

    def set_publisher(self, publisher):

        self._publisher = publisher


    def get_language(self):

        return self._language

    def set_language(self, language):

        self._language = language


    def get_year(self):

        return self._year

    def set_year(self, year):
```

```
        self._year = year


    def get_ebooks(self):
        return self._e_books


    # Method to add an e-book to the store's catalog
    def add_ebook(self, ebook):
        self._e_books.append(ebook)
        print(f"EBook '{ebook.get_title()}' added to the
store.")


    # Method to remove an e-book from the store's catalog
    def remove_ebook(self, ebook):
        if ebook in self._e_books:
            self._e_books.remove(ebook)
            print(f"EBook '{ebook.get_title()}' removed from
the store.")
        else:
            print(f"EBook '{ebook.get_title()}' not found in
the store.")
```

```
    # Method to list all e-books in the store's catalog
    def list_ebooks(self):
        return [str(ebook) for ebook in self._e_books]

    def __str__(self):
        return f"EbookStore(Category: {self._category},
Publisher: {self._publisher}, Language: {self._language},
Year: {self._year}, Total EBooks: {len(self._e_books)})"
```

Test.py

```
# test.py

from classes import EBook, ShoppingCart, Customer,
PremiumCustomer, Order, Invoice, EbookStore

def test_ebook_creation():
    # Creating some eBooks related to self-improvement and
habit-building
```

```
    ebook1 = EBook("Atomic Habits", "James Clear", "2018",
"Self-Help", 18.99)
    ebook2 = EBook("The Power of Habit", "Charles Duhigg",
"2012", "Psychology", 15.99)
    ebook3 = EBook("Tiny Habits", "BJ Fogg", "2020",
"Self-Help", 20.99)
    print(ebook1)
    print(ebook2)
    print(ebook3)
    return [ebook1, ebook2, ebook3]


def test_shopping_cart_operations():
    # Creating a shopping cart and adding/removing eBooks
    cart = ShoppingCart()
    ebooks = test_ebook_creation()


    # Adding items to the cart
    cart.add_item(ebooks[0])  # Adding "Atomic Habits"
    cart.add_item(ebooks[1])  # Adding "The Power of Habit"
    print("Shopping Cart after adding items:", cart)
```

```
    # Removing an item
    cart.remove_item(ebooks[0])  # Removing "Atomic Habits"
    print("Shopping Cart after removing an item:", cart)
    return cart


def test_customer_creation():
    # Creating a standard and a premium customer
    customer = Customer("Saif", "20222074@zu.ac.ae",
"555-1234", "Standard")
    premium_customer = PremiumCustomer("Ahmed",
"202221113@zu.ac.ae", "555-5678", "Premium",
discount_rate=0.1, loyalty_points=150)


    print(customer)
    print(premium_customer)
    return customer, premium_customer


def test_order_and_invoice():
```

```
    # Creating an order and generating an invoice for a
customer's shopping cart
    ebooks = test_ebook_creation()
    cart = ShoppingCart()

    # Adding items to the cart
    cart.add_item(ebooks[0])  # Adding "Atomic Habits"
    cart.add_item(ebooks[1])  # Adding "The Power of Habit"

    # Creating an order with a discount (e.g., premium
customer discount)
    order = Order(order_id=1, items=cart._items,
discount=0.1)  # 10% discount
    print(order)

    # Generating an invoice for the order
    invoice = Invoice(invoice_id=1001, order=order)
    print(invoice)

def test_ebookstore_operations():
```

```
    # Testing EbookStore functionality
    store = EbookStore("Self-Help", "Penguin", "English",
2023)
    ebooks = test_ebook_creation()

    # Adding e-books to the store
    store.add_ebook(ebooks[0])
    store.add_ebook(ebooks[1])
    store.add_ebook(ebooks[2])

    print("EbookStore after adding e-books:", store)
    print("List of e-books in store:", store.list_ebooks())

    # Removing an e-book
    store.remove_ebook(ebooks[1])
    print("EbookStore after removing an e-book:", store)
    print("Updated list of e-books in store:",
store.list_ebooks())

# Running the tests
```

```
print("EBook Creation:")

test_ebook_creation()



print("\nShopping Cart Operations:")

test_shopping_cart_operations()



print("\nCustomer Creation:")

test_customer_creation()



print("\nOrder and Invoice Generation:")

test_order_and_invoice()



print("\nEbookStore Operations:")

test_ebookstore_operations()
```

Github Repository:

https://github.com/saif-alh/Assignment-2---Saif.git

Summary of Learnings:

#LO1_OOAD:

I used OOAD to analyze the e-bookstore's requirements and identify key entities like Customer, EBook, Order, and Invoice. UML notations helped me map inheritance, composition, and aggregation relationships. I showed inheritance between Customer and PremiumCustomer and composition between Order and Invoice. This structured approach accurately captured real-world relationships and interactions in the system design, laying the groundwork for object-oriented code. This process helped me understand OOAD because I had to justify and explain each relationship choice.

#LO2_OOProgramming:

Building a functional and modular program required object-oriented programming. I used private attributes to summarize Python classes and give them roles and methods to follow the UML design. The classes ShoppingCart and Invoice, where I added and removed items and calculated the total with a discount and VAT, showed this. In a separate test.py file, I tested each class and method to ensure the program was error-free and met e-bookstore functional requirements. This method helped me understand OOP and build a structured, maintainable system to solve real-world problems.

#LO4_SWDocumentation:

Clear communication and documentation were my assignment priorities. Each class and method had detailed docstrings and comments explaining its purpose and functionality. I documented getter and setter methods for each class to simplify the code and used inline comments to explain complex operations like price calculations with discounts and VAT. The report template

organised and made each project section logically accessible, and the UML diagram showed entity relationships. This focus on documentation improved code readability and ensured that my design and implementation decisions were well-justified and easy to follow for future reference or modifications.