Assignment 2 Programming Fundamentals

Ahmed Waleed

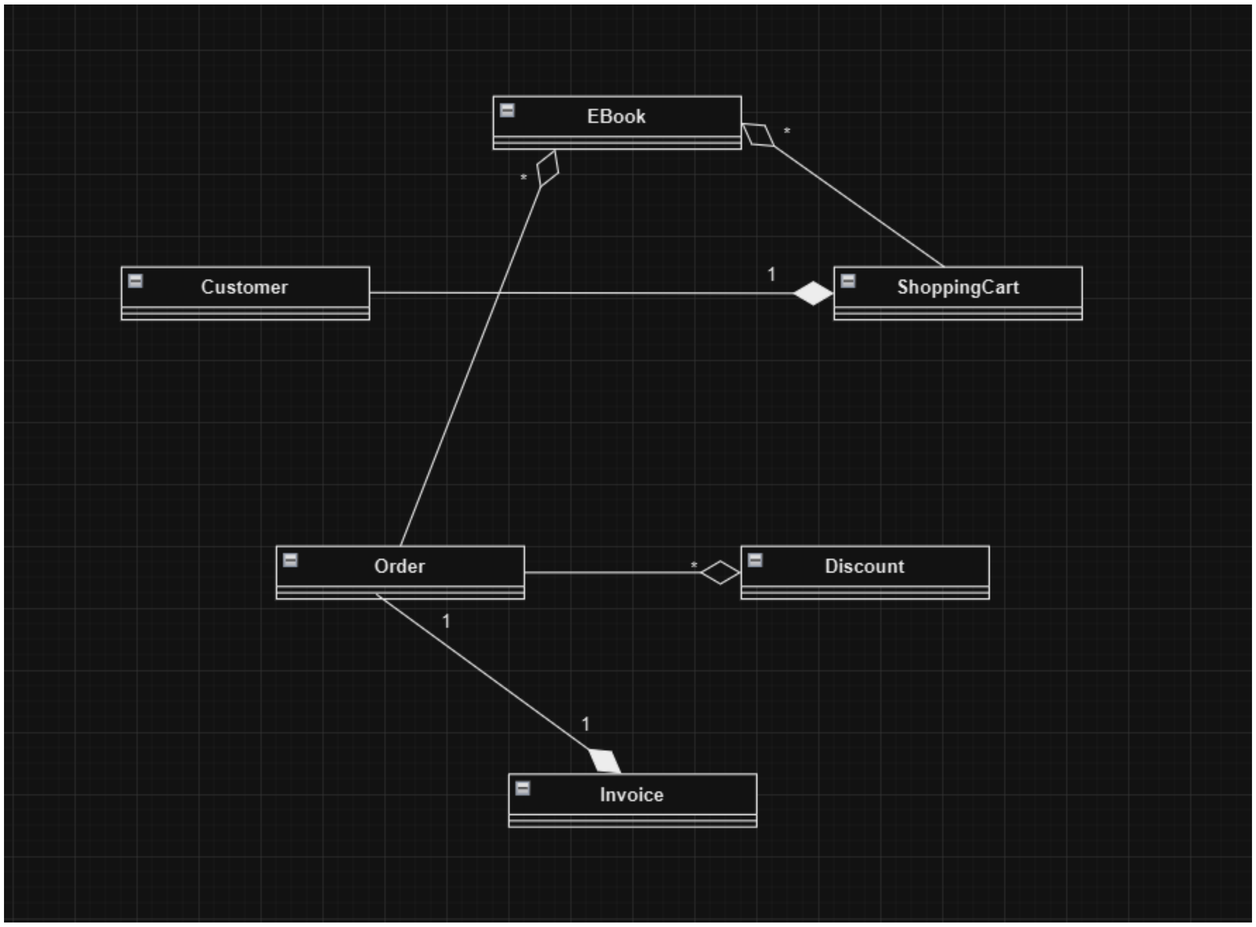
College of Interdisciplinary Studies, Zayed University

ICS220-22527: Program. Fund.

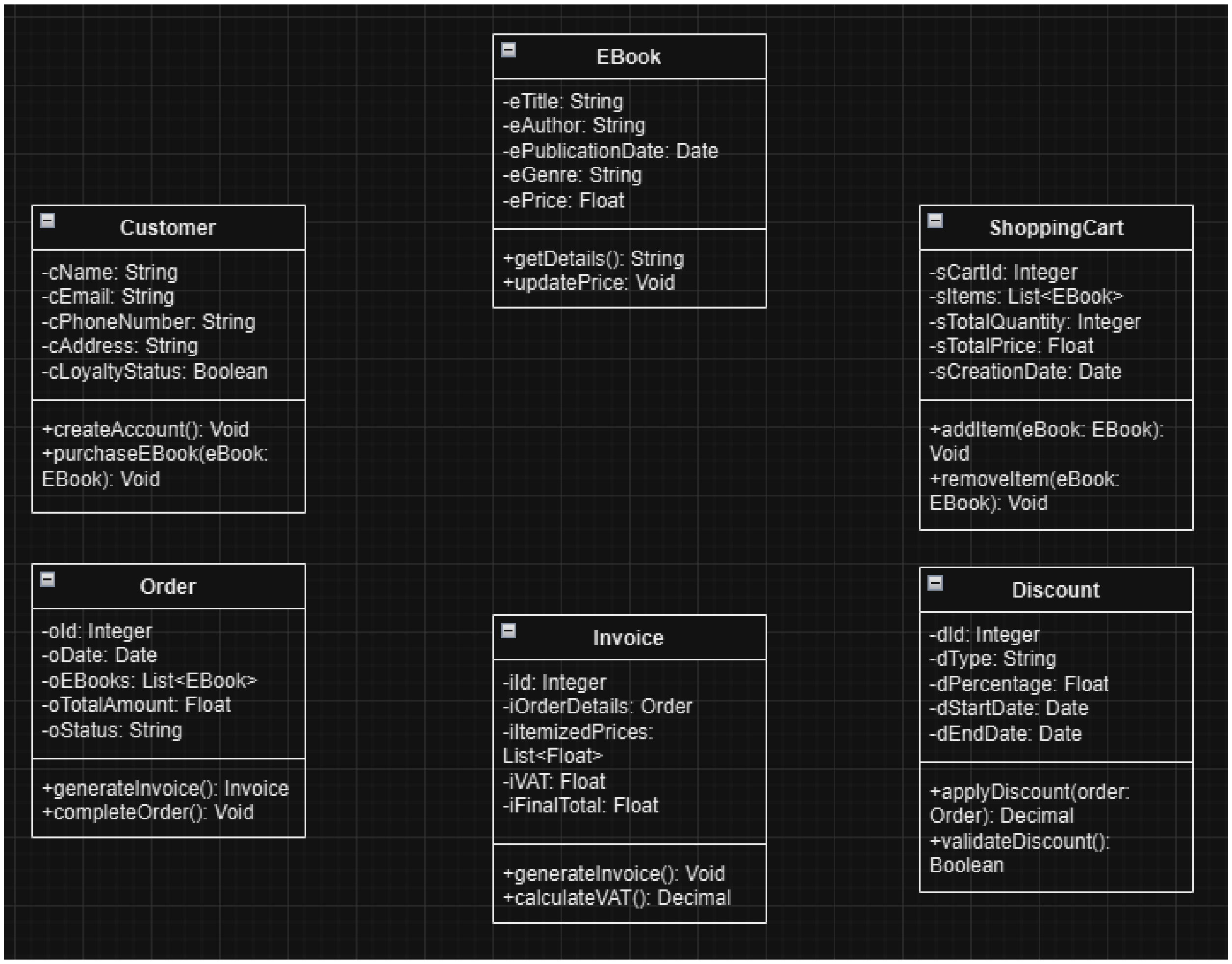
Professor: Sujith Mathew

November 4, 2024

UML Class Diagram: for the relationships



For the attributes and methods:



Description: Relationships

Relationship 1:

There is a composition relationship between the Customer class and the ShoppingCart where a Customer has a ShoppingCart that can have multiple e-books in it. Every Customer can have only one ShoppingCart, and if the Customer is removed, there will be no ShoppingCart which shows a composition.

Relationship 2:

There is an aggregation relationship between the ShoppingCart class and the EBook class where a ShoppingCart can have multiple EBooks, but still an EBook can exist independently even if the ShoppingCart was removed. The ShoppingCart is where the Ebooks are placed for them to be purchased but could be removed independently which shows aggregation.

Relationship 3:

There is an aggregation relationship between the Order class and EBook class where a Order can have multiple EBooks within the order. The EBooks are part of an order but still can exist even if the order was not there or made. So if an order was not placed, the EBooks are still going to be in the system which shows aggregation.

Relationship 4:

There is an aggregation relationship between the Order class and the Discount class where a Order can have multiple different Discounts applied to it which the system manages it throughout the payment and checking out. Discounts can still be there even if the order was removed which means that this is aggregation.

Relationship 5:

There is a composition relationship between the Order class and the Invoice class where every Order made has its own Invoice for it which has details of the order and if there were any applied discounts. An Invoice will not exist without an Order where it shows composition.

Modularity:

EBook:

This module or class manages the e-book where it has the title, author name, publishing date, and much more and can be expanded if needed.

Customer:

This module or class manages the Customer details such as his name, age, email address, and other personal things about the customer and it can be expanded if needed.

ShoppingCart:

This module or class manages the cart where it detects which e-books are added and which are removed from the cart and it calculates and shows the total for the customer where it could be modified or changed if needed.

Order:

This module or class is the main one where it manages in creating the order of the customer which is consisted of e-books, shows the total price, and shows different accessible discounts for the order.

Discount:

This module or class manages applying discounts to the orders allowing flexible promotions or loyalty benefits where it also can be modified if needed.

Invoice:

This module or class manages to create detailed invoices for the customer based on the price of the e-books he wants to purchase and shows the total paid and what discounts were applied.

Assumptions:

1. For each customer there is going to be one shopping cart each time which can help the system manage in knowing if a customer still has a cart or did he abandon it.

2. When a customer finishes purchasing the e-books, he will get them instantly after paying the invoice and will receive his order directly where the customer will not need to wait for any processing time.

3. For the VAT, all customers will have to pay 8% VAT income where it is a constant rate for all purchases made by customers.

4. Two different discounts could be applied to the order's total price where if the customer is part of the loyalty program, he is going to get a 10% discount. If he is going to purchase 5 or more e-books at the same time, he gets a 20% discount on the order.

5. One invoice is generated only after the customer has finished paying for his order which is where the invoice will show up to him. This manages how orders are completed and the customer finishes shopping.

Python Codes

EBook Class:

```
class EBook:
    """ Class representing a system that has EBooks """

    def __init__(self, title, author, publication_date, genre, price):
        self._title = title  # Title of the eBook
        self._author = author  # Author of the eBook
        self._publication_date = publication_date  # Publication date of eBook
        self._genre = genre  # Genre of the eBook
        self._price = price  # Price of the eBook

    # Setters and getters

    def set_title(self, title):
        self._title = title  # Seter for title

    def get_title(self):
        return self._title  # Geter for title

    def set_author(self, author):
        self._author = author  # Seter for author

    def get_author(self):
        return self._author  # geter for author
```

```
def set_publication_date(self, publication_date):
    self._publication_date = publication_date  # Seter for publication date

def get_publication_date(self):
    return self._publication_date  # Geter for publication date

def set_genre(self, genre):
    self._genre = genre  # Seter for genre

def get_genre(self):
    return self._genre  # Geter for genre

def set_price(self, price):
    self._price = price  # Seter for price

def get_price(self):
    return self._price  # Geter for price

def get_details(self):
    """ Returning the details of the EBook """
    return f"{self._title} by {self._author}, Genre: {self._genre}, Price: ${self._price}"

def update_price(self, new_price):
    """ Updating the price of the EBook """
    self._price = new_price

def __str__(self):
    """ Returning a string representation of the EBook """
```

```
        return f"EBook: {self._title}, Author: {self._author}, Price: ${self._price}"

# Creating the objects of the EBook class
ebook = EBook("Food", "Ahmed Alblooshi", "2023-01-05", "Life", 15.50)

# Using setter to update details
ebook.set_price(14.99)  # Update the price
ebook.set_genre("Cooking")  # Update the genre

# Printing details using the getter methods
print("Title:", ebook.get_title())
print("Author:", ebook.get_author())
print("Publication Date:", ebook.get_publication_date())
print("Genre:", ebook.get_genre())
print("Price:", ebook.get_price())

# Printing eBook details
print(ebook.get_details())

# Printing string representation of the eBook
print(ebook)
```

Customer:

```
class Customer:
    """ Class representing a customer in the system """

    def __init__(self, name, email, phone_number, address, loyalty_status):
        self._name = name  # Name of the customer
        self._email = email  # Email of the customer
        self._phone_number = phone_number  # Phone number of the customer
        self._address = address  # Address of the customer
        self._loyalty_status = loyalty_status  # Loyalty status of the customer

    # Setters and getters

    def set_name(self, name):
        self._name = name  # Setter for name

    def get_name(self):
        return self._name  # Getter for name

    def set_email(self, email):
        self._email = email  # Setter for email

    def get_email(self):
        return self._email  # Getter for email

    def set_phone_number(self, phone_number):
        self._phone_number = phone_number  # Setter for phone number

    def get_phone_number(self):
        return self._phone_number  # Getter for phone number

    def set_address(self, address):
        self._address = address  # Setter for address

    def get_address(self):
        return self._address  # Getter for address

    def set_loyalty_status(self, loyalty_status):
        self._loyalty_status = loyalty_status  # Setter for loyalty status

    def get_loyalty_status(self):
        return self._loyalty_status  # Getter for loyalty status

    def create_account(self):
        """ for account creation """
        return f"Account created for {self._name}."

    def purchase_ebook(self, ebook):
        """ for purchasing an eBook."""
        return f"{self._name} purchased the eBook '{ebook.get_title()}'."

    def __str__(self):
        """ for returning a string representation of the Customer """
        return f"Customer: {self._name}, Email: {self._email}, Loyalty Status:
{self._loyalty_status}"
```

```python
# Creating an object and instance of the Customer class
customer = Customer("Rashed", "RashedAhmed@gmail.com", "0501122345", "sheikh road", True)

# Using setter to update details
customer.set_phone_number("0554329988")  # Updating the phone number
customer.set_address("dubai st")  # Updating the address

# Printing details using the getter methods
print("Name:", customer.get_name())
print("Email:", customer.get_email())
print("Phone Number:", customer.get_phone_number())
print("Address:", customer.get_address())
print("Loyalty Status:", customer.get_loyalty_status())

# Creating an instance of EBook to demonstrate purchase_ebook method
ebook = EBook("Food", "Ahmed Alblooshi", "2022-01-05", "Cooking", 14.99)

# Printing customer actions
print(customer.create_account())
print(customer.purchase_ebook(ebook))

# Printing string representation of the Customer
print(customer)
```

ShoppingCart:

```
class ShoppingCart:
    """ Class representing a shopping cart in the system """

    def __init__(self, cart_id, creation_date):
        self._cart_id = cart_id  # ID of the shopping cart
        self._items = []  # List of EBooks in the cart
        self._total_quantity = 0  # Total number of Ebooks in the cart
        self._total_price = 0.0  # Total price of Ebooks in the cart
        self._creation_date = creation_date  # Creation date of cart

    # Setters and getters

    def set_cart_id(self, cart_id):
        self._cart_id = cart_id  # Setter for cart ID

    def get_cart_id(self):
        return self._cart_id  # Getter for cart ID

    def set_creation_date(self, creation_date):
        self._creation_date = creation_date  # Setter for creation date

    def get_creation_date(self):
        return self._creation_date  # Getter for creation date
    def add_item(self, ebook):
        """ Adds an EBook to the shopping cart """
        self._items.append(ebook)
        self._total_quantity += 1
        self._total_price += ebook.get_price()

    def remove_item(self, ebook):
        """ Removes an EBook from the shopping cart if there is any """
        if ebook in self._items:
            self._items.remove(ebook)
            self._total_quantity -= 1
            self._total_price -= ebook.get_price()

    def get_total_quantity(self):
        """ Returns the total number of EBooks in the cart """
        return self._total_quantity

    def get_total_price(self):
        """ Returns the total price of EBooks in the cart """
        return self._total_price

    def get_items(self):
        """ Returns the list of EBooks in the cart """
        return [ebook.get_title() for ebook in self._items]

    def __str__(self):
        """ Returning a string representation of the ShoppingCart """
        return f"ShoppingCart ID: {self._cart_id}, Total Quantity: {self._total_quantity}, Total 
Price: ${self._total_price}"
```

```
# Creating objects and instance of the ShoppingCart class
cart = ShoppingCart(444, "2024-06-18")

# Creating instances of EBook to add to the cart
ebook1 = EBook("Food", "Ahmed Alblooshi", "2022-01-05", "Cooking", 14.99)
ebook2 = EBook("Furniture", "Meera Ahmed", "2023-05-20", "Home", 18.50)

# Adding EBooks to the cart
cart.add_item(ebook1)
cart.add_item(ebook2)

# Printing the details using getter methods
print("Cart ID:", cart.get_cart_id())
print("Creation Date:", cart.get_creation_date())
print("Total Quantity:", cart.get_total_quantity())
print("Total Price:", cart.get_total_price())
print("Items in Cart:", cart.get_items())

# Removing an item from the cart to try
cart.remove_item(ebook1)

# Printing updated details
print("\nAfter removing an item:")
print("Total Quantity:", cart.get_total_quantity())
print("Total Price:", cart.get_total_price())
print("Items in Cart:", cart.get_items())
```

```
# Printing string representation of the ShoppingCart
print(cart)
```

Order:

```
class Order:
    """ Class representing an order in the system """

    def __init__(self, order_id, date):
        self._order_id = order_id  # ID of the order
        self._date = date  # Date of the order
        self._ebooks = []  # List of EBook objects in the order
        self._total_amount = 0.0  # Total amount for the order
        self._status = "Pending"  # Status of the order

    # Setters and getters

    def set_order_id(self, order_id):
        self._order_id = order_id  # Setter for order ID

    def get_order_id(self):
        return self._order_id  # Getter for order ID

    def set_date(self, date):
        self._date = date  # Setter for date

    def get_date(self):
        return self._date  # Getter for date


    def set_status(self, status):
        self._status = status  # Setter for status


    def get_status(self):
        return self._status  # Getter for status


    def add_ebook(self, ebook):
        """Adds an EBook object to the order and updates total amount"""
        self._ebooks.append(ebook)
        self._total_amount += ebook.get_price()


    def get_total_amount(self):
        """Returns the total amount of the order"""
        return self._total_amount


    def get_ebooks(self):
        """Returns the list of EBook objects in the order"""
        return self._ebooks


    def generate_invoice(self):
        """Generates a basic invoice for the order"""
        invoice_details = f"Order ID: {self._order_id}\nDate: {self._date}\nTotal Amount: 
${self._total_amount}\nStatus: {self._status}\nItems:\n"
        for ebook in self._ebooks:
            invoice_details += f"- {ebook.get_title()} (${ebook.get_price()})\n"
        return invoice_details
    def complete_order(self):
        """Marks the order as completed"""
        self._status = "Completed"

    def __str__(self):
        """Returning a string representation of the Order"""
        return f"Order ID: {self._order_id}, Date: {self._date}, Total Amount: 
${self._total_amount}, Status: {self._status}"

# Creating objects and an instance of the Order class
order = Order(2024, "2024-11-01")

# Creating instances of EBook to add to the order
ebook1 = EBook("Food", "Ahmed Alblooshi", "2022-01-05", "Cooking", 14.99)
ebook2 = EBook("Furniture", "Meera Ahmed", "2023-05-20", "Home", 18.50)

# Adding eBooks to the order
order.add_ebook(ebook1)
order.add_ebook(ebook2)

# Printing details using getter methods
print("Order ID:", order.get_order_id())
print("Date:", order.get_date())
print("Status:", order.get_status())
print("Total Amount:", order.get_total_amount())
print("EBooks in Order:", order.get_ebooks())

# creating and printing the invoice
print("\nInvoice Details:\n", order.generate_invoice())
```

```
# Completing the order
order.complete_order()
print("Order Status after completion:", order.get_status())

# Printing string representation of the Order
print(order)
```

Discount:

```
class Discount:
    """ Class representing a discount in the system """

    def __init__(self, discount_id, discount_type, percentage, start_date, end_date):
        self._discount_id = discount_id  # ID of the discount
        self._discount_type = discount_type  # Type of discount where there are 2 given
        self._percentage = percentage  # Discount percentage
        self._start_date = start_date  # Start date of the discount
        self._end_date = end_date  # End date of the discount

    # Setters and getters

    def set_discount_id(self, discount_id):
        self._discount_id = discount_id  # Setter for discount ID

    def get_discount_id(self):
        return self._discount_id  # Getter for discount ID

    def set_discount_type(self, discount_type):
        self._discount_type = discount_type  # Setter for discount type

    def get_discount_type(self):
        return self._discount_type  # Getter for discount type

    def set_percentage(self, percentage):
        self._percentage = percentage  # Setter for discount percentage

    def get_percentage(self):
        return self._percentage  # Getter for discount percentage

    def set_start_date(self, start_date):
        self._start_date = start_date  # Setter for start date

    def get_start_date(self):
        return self._start_date  # Getter for start date

    def set_end_date(self, end_date):
        self._end_date = end_date  # Setter for end date

    def get_end_date(self):
        return self._end_date  # Getter for end date

    def apply_discount(self, order):
        """ Applies discount to an order if it is valid and gives the discounted total """
        if self.validate_discount():
            discount_amount = order.get_total_amount() * (self._percentage / 100)
            return order.get_total_amount() - discount_amount
        return order.get_total_amount()

    def validate_discount(self):
        """ Checks if the discount is valid based on the start and end dates """
        from datetime import datetime
        current_date = datetime.now().date()
        return self._start_date <= current_date <= self._end_date

    def __str__(self):
        """ Returning a string representation of the Discount """
        return f"Discount ID: {self._discount_id}, Type: {self._discount_type}, Percentage:
{self._percentage}%"

# Creating object and an instance of the Discount class
from datetime import date
discount = Discount(554, "Loyalty", 10, date(2024, 11, 1), date(2024, 11, 30))

# Creating an instance of Order to show the discount
order = Order(554, "2024-11-01")
ebook1 = EBook("Food", "Ahmed Alblooshi", "2022-01-05", "Cooking", 14.99)
ebook2 = EBook("Furniture", "Meera Ahmed", "2023-05-20", "Home", 18.50)
order.add_ebook(ebook1)
order.add_ebook(ebook2)

# Printing discount details
print("Discount ID:", discount.get_discount_id())
print("Discount Type:", discount.get_discount_type())
print("Percentage:", discount.get_percentage())
print("Start Date:", discount.get_start_date())
print("End Date:", discount.get_end_date())

# Applying discount to the order if its valid
if discount.validate_discount():
    discounted_total = discount.apply_discount(order)
    print("Discount applied! Total after discount:", discounted_total)
else:
    print("Discount not valid. Total without discount:", order.get_total_amount())

# Printing string representation of the Discount
print(discount)
```

Invoice:

```
class Invoice:
    """ Class representing an invoice in the system """

    def __init__(self, invoice_id, order):
        self._invoice_id = invoice_id  # ID of the invoice
        self._order = order  # invoice for the order
        self._itemized_prices = [ebook.get_price() for ebook in order.get_ebooks()]  # Prices of
each eBook in the order
        self._vat = 0.05  # VAT rate 8%
        self._final_total = self.calculate_vat() + order.get_total_amount()  # Final total with VAT
    # Setters and getters

    def set_invoice_id(self, invoice_id):
        self._invoice_id = invoice_id  # Setter for invoice ID

    def get_invoice_id(self):
        return self._invoice_id  # Getter for invoice ID

    def set_order(self, order):
        self._order = order  # Setter for order

    def get_order(self):
        return self._order  # Getter for order

    def calculate_vat(self):
        """Calculates the VAT amount based on the orders total amount"""
        return self._order.get_total_amount() * self._vat

    def generate_invoice(self):
        """creates the detailed invoice with all the information"""
        invoice_details = f"Invoice ID: {self._invoice_id}\nOrder ID:
{self._order.get_order_id()}\nDate: {self._order.get_date()}\n"
        invoice_details += f"Items:\n"
        for ebook in self._order.get_ebooks():
            invoice_details += f"- {ebook.get_title()} (${ebook.get_price()})\n"
        invoice_details += f"Subtotal: ${self._order.get_total_amount():.2f}\n"
        invoice_details += f"VAT (8%): ${self.calculate_vat():.2f}\n"
        invoice_details += f"Total Amount: ${self._final_total:.2f}\n"
        return invoice_details
    def __str__(self):
        """Returning a string representation of the Invoice"""
        return f"Invoice ID: {self._invoice_id}, Order ID: {self._order.get_order_id()}, Total
Amount: ${self._final_total}"


# Creating an instance of Order to generate an invoice
order = Order(444, "2024-11-01")
ebook1 = EBook("Food", "Ahmed Alblooshi", "2022-01-05", "Cooking", 14.99)
ebook2 = EBook("Furniture", "Meera AHmed", "2023-05-20", "Home", 18.50)
order.add_ebook(ebook1)
order.add_ebook(ebook2)

# Creating objects and an instance of Invoice for the order
invoice = Invoice(554, order)

# Printing invoice details
print("Invoice ID:", invoice.get_invoice_id())
print("Order ID:", invoice.get_order().get_order_id())
print("Order Date:", invoice.get_order().get_date())
print("Invoice Total (with VAT):", invoice._final_total)

# creating and printing the full invoice details
print("\nFull Invoice Details:\n", invoice.generate_invoice())

# Printing string representation of the Invoice
print(invoice)
```

Defining Test Cases:

```
from datetime import date, datetime

class EBook:
    """ Class representing a system that has EBooks """

    def __init__(self, title, author, publication_date, genre, price):
        self._title = title  # Title of the eBook
        self._author = author  # Author of the eBook
        self._publication_date = publication_date  # Publication date of eBook
        self._genre = genre  # Genre of the eBook
        self._price = price  # Price of the eBook

    # Setters and getters

    def set_title(self, title):
        self._title = title  # Seter for title

    def get_title(self):
        return self._title  # Geter for title

    def set_author(self, author):
        self._author = author  # Seter for author

    def get_author(self):
        return self._author  # geter for author
```

```
def set_publication_date(self, publication_date):
    self._publication_date = publication_date  # Seter for publication date

def get_publication_date(self):
    return self._publication_date  # Geter for publication date

def set_genre(self, genre):
    self._genre = genre  # Seter for genre

def get_genre(self):
    return self._genre  # Geter for genre

def set_price(self, price):
    self._price = price  # Seter for price

def get_price(self):
    return self._price  # Geter for price

def get_details(self):
    """ Returning the details of the EBook """
    return f"{self._title} by {self._author}, Genre: {self._genre}, Price: ${self._price}"

def update_price(self, new_price):
    """ Updating the price of the EBook """
    self._price = new_price

def __str__(self):
        """ Returning a string representation of the EBook """
        return f"EBook: {self._title}, Author: {self._author}, Price: ${self._price}"


class Customer:
    """ Class representing a customer in the system """

    def __init__(self, name, email, phone_number, address, loyalty_status):
        self._name = name  # Name of the customer
        self._email = email  # Email of the customer
        self._phone_number = phone_number  # Phone number of the customer
        self._address = address  # Address of the customer
        self._loyalty_status = loyalty_status  # Loyalty status of the customer

    # Setters and getters

    def set_name(self, name):
        self._name = name  # Setter for name

    def get_name(self):
        return self._name  # Getter for name

    def set_email(self, email):
        self._email = email  # Setter for email

    def get_email(self):
        return self._email  # Getter for email
```

```
def set_phone_number(self, phone_number):
    self._phone_number = phone_number  # Setter for phone number

def get_phone_number(self):
    return self._phone_number  # Getter for phone number

def set_address(self, address):
    self._address = address  # Setter for address

def get_address(self):
    return self._address  # Getter for address

def set_loyalty_status(self, loyalty_status):
    self._loyalty_status = loyalty_status  # Setter for loyalty status

def get_loyalty_status(self):
    return self._loyalty_status  # Getter for loyalty status

def create_account(self):
    """ for account creation """
    return f"Account created for {self._name}."

def purchase_ebook(self, ebook):
    """ for purchasing an eBook."""
    return f"{self._name} purchased the eBook '{ebook.get_title()}'."

def __str__(self):
    """ for returning a string representation of the Customer """
```

```
        return f"Customer: {self._name}, Email: {self._email}, Loyalty Status: 
{self._loyalty_status}"


class ShoppingCart:
    """ Class representing a shopping cart in the system """

    def __init__(self, cart_id, creation_date):
        self._cart_id = cart_id  # ID of the shopping cart
        self._items = []  # List of EBooks in the cart
        self._total_quantity = 0  # Total number of Ebooks in the cart
        self._total_price = 0.0  # Total price of Ebooks in the cart
        self._creation_date = creation_date  # Creation date of cart

    # Setters and getters

    def set_cart_id(self, cart_id):
        self._cart_id = cart_id  # Setter for cart ID

    def get_cart_id(self):
        return self._cart_id  # Getter for cart ID

    def set_creation_date(self, creation_date):
        self._creation_date = creation_date  # Setter for creation date

    def get_creation_date(self):
        return self._creation_date  # Getter for creation date

    def add_item(self, ebook):
        """ Adds an EBook to the shopping cart """
        self._items.append(ebook)
        self._total_quantity += 1
        self._total_price += ebook.get_price()

    def remove_item(self, ebook):
        """ Removes an EBook from the shopping cart if there is any """
        if ebook in self._items:
            self._items.remove(ebook)
            self._total_quantity -= 1
            self._total_price -= ebook.get_price()

    def get_total_quantity(self):
        """ Returns the total number of EBooks in the cart """
        return self._total_quantity

    def get_total_price(self):
        """ Returns the total price of EBooks in the cart """
        return self._total_price

    def get_items(self):
        """ Returns the list of EBooks in the cart """
        return [ebook.get_title() for ebook in self._items]

    def __str__(self):
        """ Returning a string representation of the ShoppingCart """
        return f"ShoppingCart ID: {self._cart_id}, Total Quantity: {self._total_quantity}, Total
Price: ${self._total_price}"
```

```
class Order:
    """ Class representing an order in the system """

    def __init__(self, order_id, date):
        self._order_id = order_id  # ID of the order
        self._date = date  # Date of the order
        self._ebooks = []  # List of EBook objects in the order
        self._total_amount = 0.0  # Total amount for the order
        self._status = "Pending"  # Status of the order

    # Setters and getters

    def set_order_id(self, order_id):
        self._order_id = order_id  # Setter for order ID

    def get_order_id(self):
        return self._order_id  # Getter for order ID

    def set_date(self, date):
        self._date = date  # Setter for date

    def get_date(self):
        return self._date  # Getter for date

    def set_status(self, status):
        self._status = status  # Setter for status
    def get_status(self):
        return self._status  # Getter for status

    def add_ebook(self, ebook):
        """Adds an EBook object to the order and updates total amount"""
        self._ebooks.append(ebook)
        self._total_amount += ebook.get_price()

    def get_total_amount(self):
        """Returns the total amount of the order"""
        return self._total_amount

    def get_ebooks(self):
        """Returns the list of EBook objects in the order"""
        return self._ebooks

    def generate_invoice(self):
        """Generates a basic invoice for the order"""
        invoice_details = f"Order ID: {self._order_id}\nDate: {self._date}\nTotal Amount:
${self._total_amount}\nStatus: {self._status}\nItems:\n"
        for ebook in self._ebooks:
            invoice_details += f"- {ebook.get_title()} (${ebook.get_price()})\n"
        return invoice_details

    def complete_order(self):
        """Marks the order as completed"""
        self._status = "Completed"

    def __str__(self):
        """Returning a string representation of the Order"""
        return f"Order ID: {self._order_id}, Date: {self._date}, Total Amount:
${self._total_amount}, Status: {self._status}"



class Discount:
    """ Class representing a discount in the system """

    def __init__(self, discount_id, discount_type, percentage, start_date, end_date):
        self._discount_id = discount_id  # ID of the discount
        self._discount_type = discount_type  # Type of discount where there are 2 given
        self._percentage = percentage  # Discount percentage
        self._start_date = start_date  # Start date of the discount
        self._end_date = end_date  # End date of the discount

    # Setters and getters

    def set_discount_id(self, discount_id):
        self._discount_id = discount_id  # Setter for discount ID

    def get_discount_id(self):
        return self._discount_id  # Getter for discount ID

    def set_discount_type(self, discount_type):
        self._discount_type = discount_type  # Setter for discount type

    def get_discount_type(self):
        return self._discount_type  # Getter for discount type
```

```
def set_percentage(self, percentage):
    self._percentage = percentage  # Setter for discount percentage

def get_percentage(self):
    return self._percentage  # Getter for discount percentage

def set_start_date(self, start_date):
    self._start_date = start_date  # Setter for start date

def get_start_date(self):
    return self._start_date  # Getter for start date

def set_end_date(self, end_date):
    self._end_date = end_date  # Setter for end date

def get_end_date(self):
    return self._end_date  # Getter for end date

def apply_discount(self, order):
    """ Applies discount to an order if it is valid and gives the discounted total """
    if self.validate_discount():
        discount_amount = order.get_total_amount() * (self._percentage / 100)
        return order.get_total_amount() - discount_amount
    return order.get_total_amount()

def validate_discount(self):
    """ Checks if the discount is valid based on the start and end dates """
    current_date = datetime.now().date()
```

```
        return self._start_date <= current_date <= self._end_date


    def __str__(self):
        """ Returning a string representation of the Discount """
        return f"Discount ID: {self._discount_id}, Type: {self._discount_type}, Percentage: 
{self._percentage}%"



class Invoice:
    """ Class representing an invoice in the system """


    def __init__(self, invoice_id, order):
        self._invoice_id = invoice_id  # ID of the invoice
        self._order = order  # invoice for the order
        self._itemized_prices = [ebook.get_price() for ebook in order.get_ebooks()]  # Prices of 
each eBook in the order
        self._vat = 0.05  # VAT rate 8%
        self._final_total = self.calculate_vat() + order.get_total_amount()  # Final total with VAT


    # Setters and getters


    def set_invoice_id(self, invoice_id):
        self._invoice_id = invoice_id  # Setter for invoice ID


    def get_invoice_id(self):
        return self._invoice_id  # Getter for invoice ID


    def set_order(self, order):
        self._order = order  # Setter for order
    def get_order(self):
        return self._order  # Getter for order

    def calculate_vat(self):
        """Calculates the VAT amount based on the orders total amount"""
        return self._order.get_total_amount() * self._vat

    def generate_invoice(self):
        """creates the detailed invoice with all the information"""
        invoice_details = f"Invoice ID: {self._invoice_id}\nOrder ID: 
{self._order.get_order_id()}\nDate: {self._order.get_date()}\n"
        invoice_details += f"Items:\n"
        for ebook in self._order.get_ebooks():
            invoice_details += f"- {ebook.get_title()} (${ebook.get_price()})\n"
        invoice_details += f"Subtotal: ${self._order.get_total_amount():.2f}\n"
        invoice_details += f"VAT (5%): ${self.calculate_vat():.2f}\n"
        invoice_details += f"Total Amount: ${self._final_total:.2f}\n"
        return invoice_details

    def __str__(self):
        """Returning a string representation of the Invoice"""
        return f"Invoice ID: {self._invoice_id}, Order ID: {self._order.get_order_id()}, Total 
Amount: ${self._final_total}"



# Test Cases
print("Test Case: EBook Operations")
ebook = EBook("Food", "Ahmed Alblooshi", "2023-01-05", "Life", 15.50)
```

```
ebook.set_price(14.99)
ebook.set_genre("Cooking")
print(ebook)


print("Test Case: Customer Account")
customer = Customer("Rashed", "RashedAhmed@gmail.com", "0501122345", "sheikh road",
True)
print(customer)
print(customer.create_account())
print(customer.purchase_ebook(ebook))


print("Test Case: Add/Modify/Remove Customer Account")


# Adding a new customer
customer = Customer("Rashed", "RashedAhmed@gmail.com", "0501122345", "Sheikh Road",
True)
print("Customer added:", customer)


# Modifying customer details
customer.set_phone_number("0554329988")  # Update phone number
customer.set_address("Dubai St")  # Update address
print("Customer details after modification:", customer)


# Removing customer
del customer
print("Customer removed.")



print("Test Case: Shopping Cart")
```

```
cart = ShoppingCart(444, "2024-06-18")
cart.add_item(ebook)
ebook2 = EBook("Furniture", "Meera Ahmed", "2023-05-20", "Home", 18.50)
cart.add_item(ebook2)
print(cart)
cart.remove_item(ebook)
print(cart)

print("Test Case: Order with Discount")
discount = Discount(554, "Loyalty", 10, date(2024, 11, 1), date(2024, 11, 30))
order = Order(554, "2024-11-01")
order.add_ebook(ebook)
order.add_ebook(ebook2)
if discount.validate_discount():
    print("Discount applied! Total after discount:", discount.apply_discount(order))

print("Test Case: Invoice Generation")
invoice = Invoice(554, order)
print(invoice.generate_invoice())
print(invoice)
```