Software Implementation - OOP Project

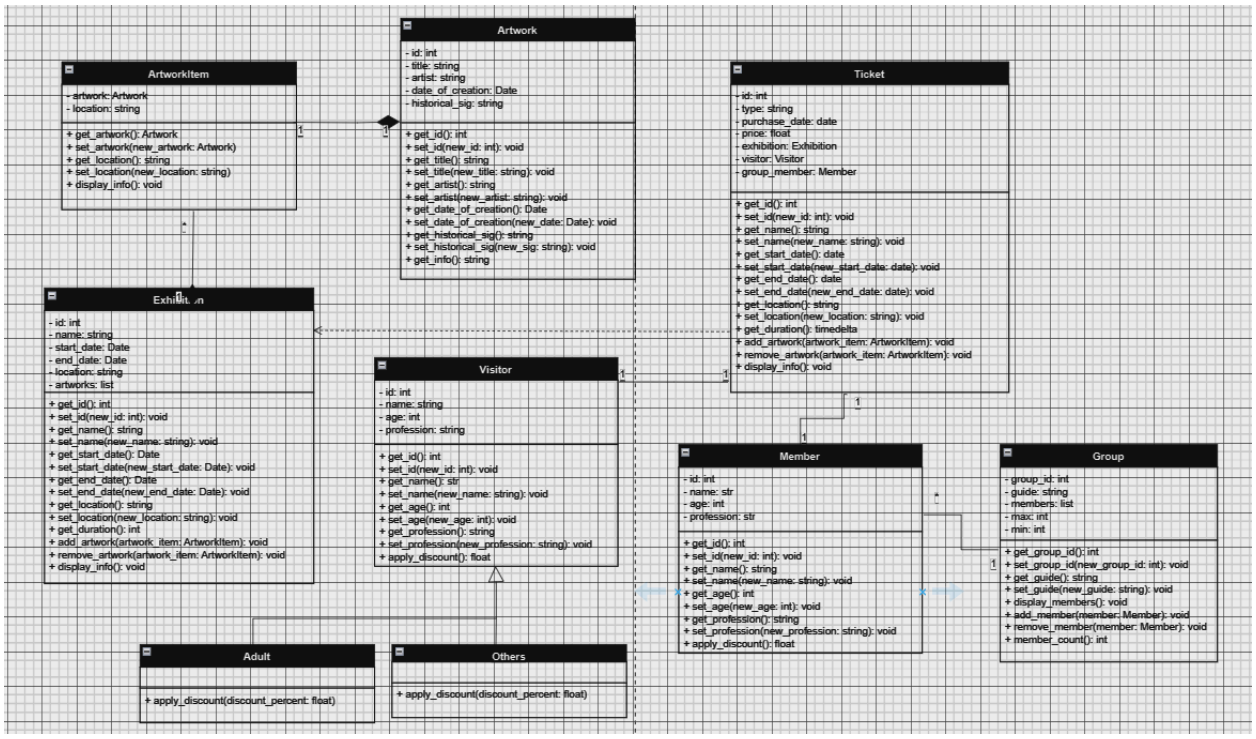
Report

Student Name:

Student ID:

Submission Date:

UML Class Diagram



There are nine classes in the diagram: Artwork, ArtworkItem, Ticket, Exhibition, Visitor, Member, Group, Adult, and Others. Artwork contains attributes like title, artist, date of creation, and historical significance and access methods for them. Artwork contains an instance of Artwork and stores the location of the artwork. In addition, to the setters and getters, the class contains a display function to display the art information of both classes. The Artwork and ArtworkItam classes do a single purpose of storing artwork and artifacts for the museum. It means that if the Artwork class is deleted ArtworkItem serves no purpose at all.

Then comes the Exhibition class which is where the artworks from the museum are displayed. An exhibition can be permanent galleries, exhibition halls, and outdoor spaces. It encapsulates essential attributes such as an ID for identification, a name to describe the exhibition, start and end dates to define its duration, a location where it takes place, and a list of artworks included in the exhibition. This class enables the management and organization of

exhibitions, allowing for the addition and removal of artworks, as well as providing functionality to display information about the exhibition, such as its ID, name, dates, location, and the artworks featured in it. ArtworkItem and Exhibition have an association relation. 1 artwork item can be displayed in 1 exhibition whereas 1 exhibition can have multiple artwork items.

An exhibition or event ticket for entry into a museum or other cultural facility is represented by the Ticket class. A ticket's qualities include identity information such as an ID, the type of ticket (e.g. online, in-person), the price, the date of purchase, information about the exhibition or event for which the ticket is intended, the visitor or group member who is purchasing the ticket, and the group member, if applicable. The course covers techniques for verifying the ticket using the date of the exhibition, computing the ticket cost with any relevant savings, and presenting ticket facts such as ID, purchase date, cost, related guest or group member, and exhibition or event specifics. The connection between the Exhibition and Ticket classes is that a ticket is linked to an exhibition and serves as a pass for entry or access to the event or exhibition within a museum or other cultural institution. Specifically, the Ticket is dependent on the Exhibition class as indicated by the arrow in the UML diagram.

The Visitor class represents a generic visitor to a museum or cultural institution and includes attributes such as ID, name, age, and profession. The Others and Adult classes are subclasses of Visitor and inherit from it. Each of the Others and Adult classes has a specialized discount logic for ticket pricing based on age and profession. The Others class provides free tickets for visitors under 18 or over 60 years old, as well as for students and teachers. In contrast, the Adult class applies no discounts for visitors aged 18 to 60. The relation between the Visitor and Ticket classes is that a ticket can be associated with a visitor (1 to 1 relation) when

purchased, indicating which visitor or group member has bought the ticket and allowing for specific discount calculations based on the visitor's age and profession or group membership.

A group of visitors in a museum is represented by the `Group` class, which has members, a list of members, and a group ID. A single visitor within a group is represented by the Member class, which has attributes like ID, name, age, and occupation. A minimum and maximum member limit is upheld by the Group class, which controls member addition and removal. 1 member has 1 group and 1 group has many members. The Member and Ticket classes are related in that when a ticket is purchased in a group setting, it can be linked to a member, identifying which group member purchased the ticket. These two classes have an association relation. 1 member has 1 ticket and vice versa.

Python Code

Following are the Python classes implementation of the above UML diagram:

1. Artwork

```
class Artwork:
    def __init__(self,  id, title, artist, date_of_creation, historical_sig):
        self.__id = id
        self.__title = title
        self.__artist = artist
        self.__date_of_creation = date_of_creation
        self.__historical_sig = historical_sig

    def get_id(self):
        return self.__id
    def set_id(self, new_id):
        self.__id = new_id
    def get_title(self):
        return self.__title
    def set_title(self, new_title):
        self.__title = new_title
    def get_artist(self):
        return self.__artist
    def set_artist(self, new_artist):
        self.__artist = new_artist
    def get_date_of_creation(self):
        return self.__date_of_creation
```

```
    def set_date_of_creation(self, new_date):
        self.__date_of_creation = new_date
    def get_historical_sig(self):
        return self.__historical_sig
    def set_historical_sig(self, new_sig):
        self.__historical_sig = new_sig

    def get_info(self):
        return f"ID: {self.__id}\nTitle: {self.__title}\nArtist: {self.__artist}\nDate of
Creation: {self.__date_of_creation}\nHistorical Significance: {self.__historical_sig}"
```

2. ArtworkItem

```
class ArtworkItem:
    def __init__(self, artwork, location):
        self.__artwork = artwork
        self.__location = location

    def get_artwork(self):
        return self.__artwork
    def set_artwork(self, new_artwork):
        self.__artwork = new_artwork
    def get_location(self):
        return self.__location
    def set_location(self, new_location):
        self.__location = new_location

    def display_info(self):
        print("--Artwork--")
        print(self.__artwork.get_info())
        print(f"Location: {self.__location}")
```

3. Exhibition

```
class Exhibition:
    def __init__(self, id, name, start_date, end_date, location):
        self.__id = id
        self.__name = name
        self.__start_date = start_date
        self.__end_date = end_date
        self.__location = location
        self.__artworks = [] #creating a list to store the diffreent artworks in an exhibition

    def get_id(self):
        return self.__id
    def set_id(self, new_id):
        self.__id = new_id
```

```
    def get_name(self):
        return self.__name
    def set_name(self, new_name):
        self.__name = new_name
    def get_start_date(self):
        return self.__start_date
    def set_start_date(self, new_start_date):
        self.__start_date = new_start_date
    def get_end_date(self):
        return self.__end_date
    def set_end_date(self, new_end_date):
        self.__end_date = new_end_date
    def get_location(self):
        return self.__location
    def set_location(self, new_location):
        self.__location = new_location

    def get_duration(self):
        return self.__end_date - self.__start_date

    def add_artwork(self, artwork_item):
        self.__artworks.append(artwork_item) #adding artwork in list
    def remove_artwork(self, artwork_item):
        self.__artworks.remove(artwork_item) #removing artwork from list

    def display_info(self):
        print("--Exhibition--")
        print("ID: ", self.__id)
        print("Name: ", self.__name)
        print("Start Date: ", self.__start_date)
        print("End Date: ", self.__end_date)
        print("Location: ", self.__location)
        print("Artworks in Exhibition: ", end=" ")
        for artwork_item in self.__artworks:
            print(artwork_item.get_artwork().get_title(), end="   ")
        print()
```

This class contains a get_duration() function that returns the subtracted result from the end date and start date of the exhibition. The class contains a list that stores all artworks and contains an add and remove artwork function as well.

4. Visitor

```
class Visitor:
    def __init__(self, id, name, age, profession):
        self.__id = id
        self.__name = name
```

```
        self.__age = age
        self.__profession = profession

    def get_id(self):
        return self.__id
    def set_id(self, new_id):
        self.__id = new_id
    def get_name(self):
        return self.__name
    def set_name(self, new_name):
        self.__name = new_name
    def get_age(self):
        return self.__age
    def set_age(self, new_age):
        self.__age = new_age
    def get_profession(self):
        return self.__profession
    def set_profession(self, new_profession):
        self.__profession = new_profession

    def apply_discount(self):
        return 0
```

5. Adult

```
class Adult(Visitor):
    def apply_discount(self):
        if 18 <= self.get_age() <= 60:
            return 0  #no discount
        else:
            return -1  #invalid age for adult
```

6. Others

```
class Others(Visitor):
    def apply_discount(self):
        if self.get_age() <18 or self.get_age() >60 or self.get_profession() == "student" or
self.get_profession() == "teacher":
            return 1 #free ticket
        else:
            return 0 #no discount
```

The Others and Afult classes inherit from Visitor. They are created to handle the discounting process for these 2 groups. Adults between 18 and 60 are supposed to pay the full

ticket price of 63 AED whereas visitors under 18 (young people) or over 60 (seniors) years old and students and teachers are all given a 100 percent discount, which means the final ticket price for them is zero.

7. Group

```
class Group:
    def __init__(self, group_id, guide):
        self.__group_id = group_id
        self.__guide = guide
        self.__members = [] #members in a group
        if len(self.__members) < 15:
            print()
            print("--Error!!: Cannot create group with less than 15 members--")
            print()
        self.__max = 40
        self.__min = 15

    def get_group_id(self):
        return self.__group_id
    def set_group_id(self, new_group_id):
        self.__group_id = new_group_id
    def get_guide(self):
        return self.__guide
    def set_guide(self, new_guide):
        self.__guide = new_guide

    def display_members(self):
        print("Group Members: " , end=" ")
        for i in self.__members:
            print(i.get_name(), end="     ")
        print()

    def add_member(self, member):
        if self.member_count() <= self.__max:
            self.__members.append(member)

    def remove_member(self, member):
        if member in self.__members and self.member_count() >= self.__min:
            self.__members.remove(member)
        else:
            print(f"{member} is not a member of this group.")

    def member_count(self):
        return len(self.__members) #count of mmebers
```

The Group class has a maximum and minimum member limit as well. The add_member() and remove_member() are catered to deal with this. Then is the member_count() function that calculates the count of group members. A guide is also used as the leader of the group. Moreover, a display message is presented if a group is created for members less than forty. The message goes away as soon as the member count exceeds 15.

8. Member

```
class Member:
    def __init__(self, id, name, age, profession):
        self.__id = id
        self.__name = name
        self.__age = age
        self.__profession = profession

    def get_id(self):
        return self.__id
    def set_id(self, new_id):
        self.__id = new_id
    def get_name(self):
        return self.__name
    def set_name(self, new_name):
        self.__name = new_name
    def get_age(self):
        return self.__age
    def set_age(self, new_age):
        self.__age = new_age
    def get_profession(self):
        return self.__profession
    def set_profession(self, new_profession):
        self.__profession = new_profession

    def apply_discount(self):
        return 0.5  #50% discount
```

Each member of a group is given a 50 percent.

9. Ticket

```
class Ticket:
    def __init__(self, id, type, purchase_date, price, exhibition, visitor, group_member):
        self.__id = id
        self.__type = type
        self.__purchase_date = purchase_date
```

```
        self.__price = price
        self.__exhibition = exhibition
        self.__visitor = visitor
        self.__group_member = group_member

    def get_id(self):
        return self.__id
    def set_id(self, new_id):
        self.__id = new_id
    def get_type(self):
        return self.__type
    def set_type(self, new_type):
        self.__type = new_type
    def get_purchase_date(self):
        return self.__purchase_date
    def set_purchase_date(self, new_purchase_date):
        self.__purchase_date = new_purchase_date
    def get_price(self):
        return self.__price
    def set_price(self, new_price):
        self.__price = new_price
    def get_exhibition(self):
        return self.__exhibition
    def set_exhibition(self, new_exhibition):
        self.__exhibition = new_exhibition
    def get_visitor(self):
        return self.__visitor
    def set_visitor(self, new_visitor):
        self.__visitor = new_visitor
    def get_group_member(self):
        return self.__group_member
    def set_group_member(self, new_group_member):
        self.__group_member = new_group_member

    def validate_ticket(self):
        current = date.today()
                    if current >= self.__exhibition.get_start_date() and current <=
self.__exhibition.get_end_date():
            return True
        else:
            return False

    def calculate_price(self):

        if self.__visitor != None:
            discount_percent = self.__visitor.apply_discount()
        else:
            discount_percent = self.__group_member.apply_discount()

        if discount_percent == -1:
            return None
```

```
        elif isinstance(self.__group_member, Member):  #group member
            discount_percent = self.__group_member.apply_discount()
        elif isinstance(self.__visitor, Visitor):  #visitor
            discount_percent = self.__visitor.apply_discount()

        discounted_price = self.__price - (self.__price * discount_percent)
        final_price = discounted_price * 1.05  # 5% VAT

        return final_price

    def display_info(self):
        print("--Ticket--")
        print("ID: ",self.__id)
        print("Purchase Date: ", self.__purchase_date)
        print("Price: ", self.__price)

        if self.__visitor != None:
            print("Visitor Name: ",self.__visitor.get_name())
            print("Visitor ID: ", self.__visitor.get_id())
        if self.__group_member != None:
            print("Group Member Name: ", self.__group_member.get_name())
            print("Group Member ID: ", self.__group_member.get_id())

        print("Exhibition Location: ", self.__exhibition.get_location()) #exhibition location
        print("Exhibition Name: ", self.__exhibition.get_name()) #name of exhibition
        print("Exhibition Duration: ", self.__exhibition.get_duration()) #time of exhibition
```

By comparing the current date with the beginning and ending dates of the associated exhibition, the validate_ticket() method determines whether the ticket is still valid. If the ticket is legitimate, it returns True; if not, it returns False. The final ticket price is determined by the calculate_price() method, which takes into account any applicable discounts based on the visitor or group member. It initially determines whether the ticket is linked to a visitor; if not, it applies the group member's discount. Calling the visitor's or group member's apply_discount() method yields the discount. It computes the discounted price and adds 5% VAT to get the final price if the discount is valid (a value of -1 indicates no discount).

Test Cases

1.

```
############(a) addition of new art############
```

```
new_artwork = Artwork(1, "Starry Night", "Vincent van Gogh", date(1889, 1, 1),
"Post-Impressionist masterpiece")
new_artwork_item = ArtworkItem(new_artwork, "Gallery B")
new_artwork_item.display_info()
print()
new_artwork1 = Artwork(2, "Mona Lisa", "Leonardo da Vinci", date(1503, 1, 1), "Iconic
portrait")
new_artwork_item1 = ArtworkItem(new_artwork1, "Gallery B")
new_artwork_item1.display_info()
print()
```

Result

```
--Artwork--
ID: 1
Title: Starry Night
Artist: Vincent van Gogh
Date of Creation: 1889-01-01
Historical Significance: Post-Impressionist masterpiece
Location: Gallery B

--Artwork--
ID: 2
Title: Mona Lisa
Artist: Leonardo da Vinci
Date of Creation: 1503-01-01
Historical Significance: Iconic portrait
Location: Gallery B
```

2.

```
############(b) opening a new museum exhibition############
new_exhibition = Exhibition(1, "Impressionist Masterpieces", date(2024, 7, 1), date(2024,
9, 30), "East Wing")
#adding artworks to exhibition
new_exhibition.add_artwork(new_artwork_item)
new_exhibition.add_artwork(new_artwork_item1)
new_exhibition.display_info()
print()
```

Result

```
--Exhibition--
ID:  1
Name:  Impressionist Masterpieces
Start Date:  2024-07-01
End Date:  2024-09-30
Location:  East Wing
Artworks in Exhibition:  Starry Night   Mona Lisa
```

3.

```
############(c) purchase of ticket############
#adult purchases ticket
adult_visitor = Adult(1, "Alice Johnson", 25, "engineer")
adult_ticket = Ticket(1, "Inperson", date.today(), 63, new_exhibition, adult_visitor, None)

#student purchases ticket
student_visitor = Others(1, "Adam Newman", 20, "student")
student_ticket = Ticket(2, "Online", date.today(), 63, new_exhibition, student_visitor,
None)

#group purchases ticket
group1 = Group(1, "tour guide")
member1 = Member(1, "Joden Dick", 30, "content creator")
member2 = Member(2, "Jane Smith", 25, "doctor")
group1.add_member(member1)
group1.add_member(member2)
member_ticket1 = Ticket(3, "Inperson", date.today(), 63, new_exhibition, None, member1)
member_ticket2 = Ticket(4, "Online", date.today(), 63, new_exhibition, None, member2)
```

Result

```
--Error!!: Cannot create group with less than 15 members--
```

The error appears because as of now the group members are lower than 15, the min limit.

4.

```
############(d) ticket reciepts + final price############
adult_ticket.display_info()
print("ticket final price: ",adult_ticket.calculate_price())
```

```
print()
student_ticket.display_info()
print("ticket final price: ",student_ticket.calculate_price())
print()
member_ticket1.display_info()
print("ticket final price: ",member_ticket1.calculate_price())
print()
member_ticket2.display_info()
print("ticket final price: ",member_ticket2.calculate_price())
```

Result

```
--Ticket--
ID:  1
Purchase Date:  2024-03-16
Price:  63
Visitor Name:  Alice Johnson
Visitor ID:  1
Exhibition Location:  East Wing
Exhibition Name:  Impressionist Masterpieces
Exhibition Duration:  91 days, 0:00:00
ticket final price:  66.15

--Ticket--
ID:  2
Purchase Date:  2024-03-16
Price:  63
Visitor Name:  Adam Newman
Visitor ID:  1
Exhibition Location:  East Wing
Exhibition Name:  Impressionist Masterpieces
Exhibition Duration:  91 days, 0:00:00
ticket final price:  0.0
```

```
--Ticket--
ID:  3
Purchase Date:  2024-03-16
Price:  63
Group Member Name:  Joden Dick
Group Member ID:  1
Exhibition Location:  East Wing
Exhibition Name:  Impressionist Masterpieces
Exhibition Duration:  91 days, 0:00:00
ticket final price:  33.075

--Ticket--
ID:  4
Purchase Date:  2024-03-16
Price:  63
Group Member Name:  Jane Smith
Group Member ID:  2
Exhibition Location:  East Wing
Exhibition Name:  Impressionist Masterpieces
Exhibition Duration:  91 days, 0:00:00
ticket final price:  33.075
PS C:\Users\Dell>
```

Github Repository Link

https://github.com/beg-python/artwork

Summary of Learnings

Through this project, I have gained valuable insights into object-oriented programming (OOP) principles and software design practices. One of the key learnings was the importance of encapsulation in OOP. Encapsulation allowed me to hide the internal details of classes, such as attributes and methods, and expose only the necessary functionalities through well-defined interfaces. I learned that this not only enhances the modularity of the code but also promotes code reusability.

Making the UML diagram also made it easier for me to see how various classes relate to one another and to comprehend how data moves through the system and interacts. In order to properly simulate real-world entities, it stressed the importance of creating classes with distinct

roles and defining suitable interactions, such as affiliations, inheritance, and dependencies. My comprehension of software design concepts like as composition, polymorphism, inheritance, and abstraction has improved as a result of this training, and I am now able to create software that is more structured, arranged, and scalable. All things considered, this project gave me the chance to put the theoretical ideas of object-oriented programming and software design into practice in a real-world setting, which improved my system design and programming abilities. Also, the process of doing the process in recurring steps gave me an opportunity to repeat the code and find flaws that existed. This helped me in avoiding them the next time.