Assignment 1: Software Modelling - UML Use Case and UML Class diagrams

ICS220 – Programming Fundamental

Professor Mohammad Kuhail

Maryam Alwars

Greetings from the **Garage Service System** a system that aids in controlling a car garage's operations. This system is intended for appointment scheduling, repairing vehicles, and generating invoices for the work done. Customers, the appointment and billing team, the service team, and the garage system are the actors that can use the garage service system.

<u>Use cases :</u>

Create an account: In this use case, a new customer can register for a car garage account by giving their personal information, and contact information, including first name, last name, email, phone number, and address. The customer can manage their vehicles after creating an account.

Manage appointments: This use case enables customers to make, modify, or cancel appointments. Customers can examine various appointment slots and choose the one that best suits their schedules. In order to guarantee that only users with permission can manage appointments, the use case additionally incorporates customer authentication.

Repair the vehicle: This use case represents the work done by the service team in the car garage system to repair the customer's car. The use case involves recording information about the vehicle, making repairs, and updating the vehicle's condition in the database.

Generate invoice: With this use case, the appointment and billing team is able to create an invoice for the customer based on the vehicle repairs that were carried out. The use case includes figuring out the total price, applying any discounts that apply, and informing the customer of the total amount. A discount extension that can be applied to the invoice is also extended from the use case.

Actors:

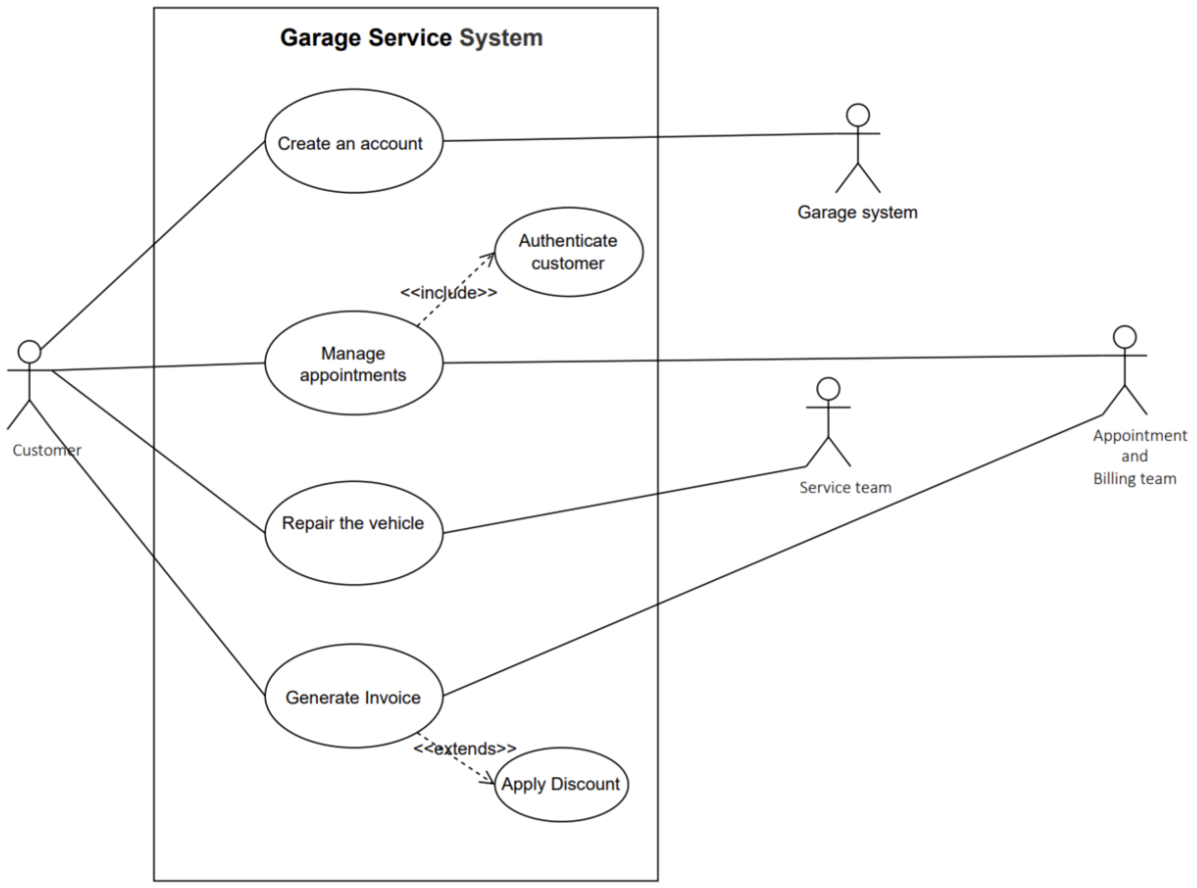
Customer: is one of the actors in the Car Garage System and interacts with it to create an account, make appointments, manage vehicle repairs, and asks for invoices. The Car Garage System helps customers have a satisfying experience, which increases customer satisfaction and loyalty and can result in more sales and repeat business.

Appointment and billing team: is one of the actors in the Car Garage System. This team is in charge of scheduling appointments and creating invoices for customers. The team makes appointments and creates invoices based on the work done on the customer's vehicle using the car garage system.

Service team: is one of the actors in the Car Garage System. This team is in charge of repairing the customer's vehicle. While repairing the vehicle the team can update the vehicle's status.

Garage system: is one of the actors in the Car Garage System. The system is in charge of storing customer information such as personal information, contact information, username and pin code.

UML Use-Case Diagram



UML Use-Case Description

Use Case:	Create an account
Trigger:	The user wants to register into the garage system
Main Scenario:	
1.	The customer asks to register as a new customer in the garage system
2.	The customer enters his personal details
3.	The customer enters his contact information
4.	The garage system verifies the contact information
5.	The garage system verifies save the user's information into the database
6.	The garage system asks the user to create a username and pin code
7.	The garage system verifies the username and pin code
8.	The customer logs in to the garage system using the username and pin code
9.	The customer adds the vehicle details to the account
10.	The garage system saves the vehicle information

Exceptions:	
4a.	1. The contact information is wrong. 2. The garage system notifies the customer and asks the customer to enter the correct contact information.
6a.	1. The username is already used. 2. The garage system notifies the customer and asks the customer to choose another username.
8a.	1. The username and pin code does not match. 2. The garage system notifies the customer and asks the customer to enter the right username and password.

Use Case:	Manage appointments
Trigger:	The customer wants to take an appointment
Preconditions:	The customer is authenticated
Main Scenario:	
1.	The customer asks the appointment and billing team to take an appointment.
2.	The appointment and billing team asks the customer to provide the username and pin code.
3.	The customer provides the username and pin code to the appointment and billing team.
4.	The customer selects a vehicle.
5.	The customer selects the services.
6.	The customer selects a Date and time.
7.	The appointment and billing team verifies the availability
8.	The appointment and billing team confirms the appointment to the customer.
Exceptions:	
3a.	1. The username and pin code does not match. 2. The appointment and billing team notifies the customer and asks to enter the right username and pin code.
4a.	1. The vehicle is not registered into the garage system. 2. The appointment and billing team notifies the customer that the vehicle need to be registered into the garage system.
7a.	1. There is no availability in the selected date and time. 2. The appointment and billing team asks the customer to change the selected data and time.

Use Case:	Repair the vehicle
Trigger:	The customer wants to repair the vehicle
Preconditions:	The customer has an appointment
Main Scenario:	
1.	The customer drops the vehicle at the garage.

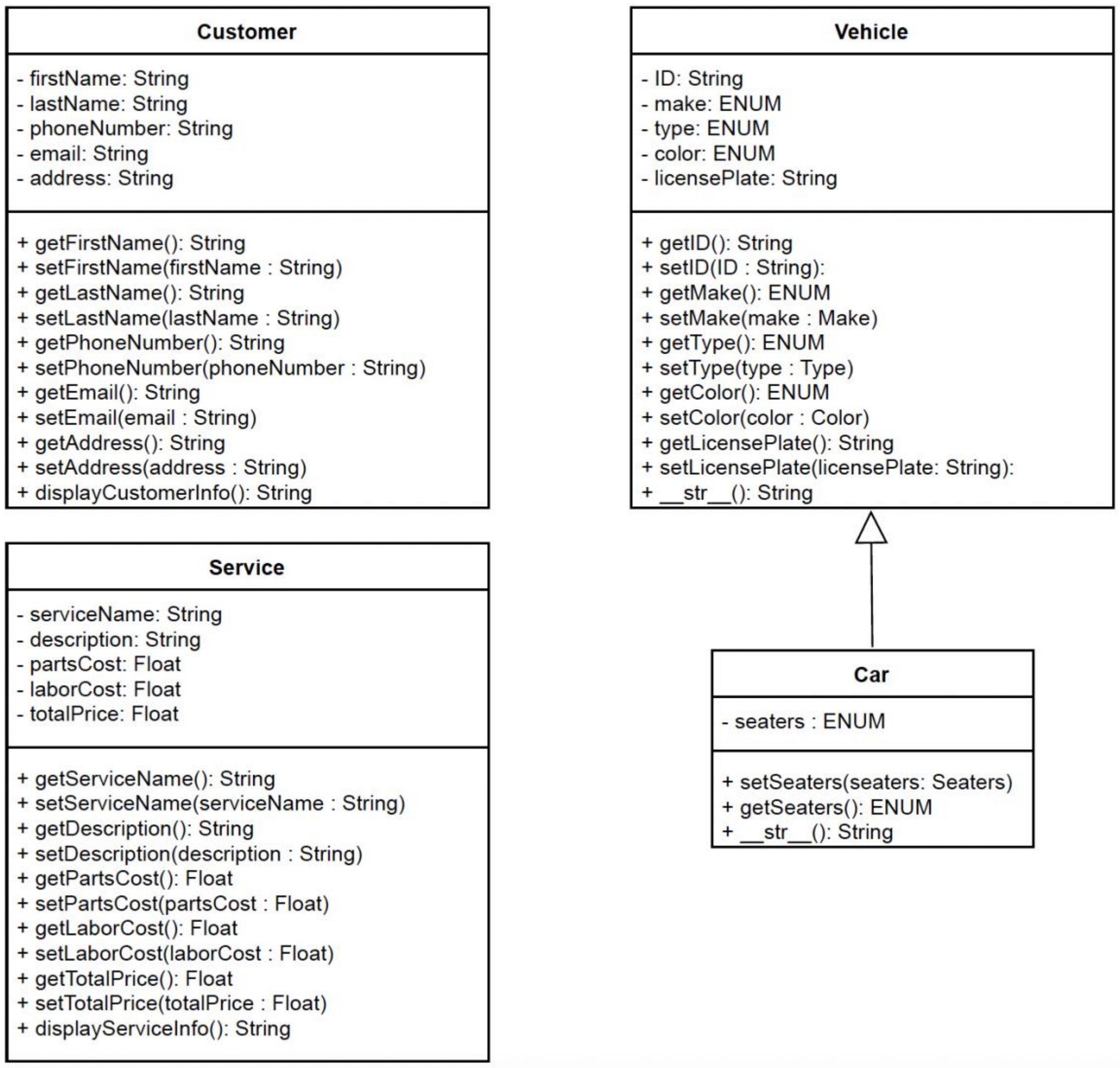
2.	The service team verifies that the customer has an appointment on the given data and time.
3.	The service team receives the vehicle from the customer.
4.	The service team updates the state of the service to “Under process”.
5.	The service team repairs the vehicle.
6.	The service team updates the state of the service to “Repaired and ready”.
Exceptions:	
1a.	1. The customer did not drop the vehicle at the garage. 2. The service team asks the appointment and billing team to contact the customer and reschedule the appointment.
2a.	1. The customer does not have an appointment. 2. The service team asks the customer to schedule an appointment. 3. Use case is terminated. 4.

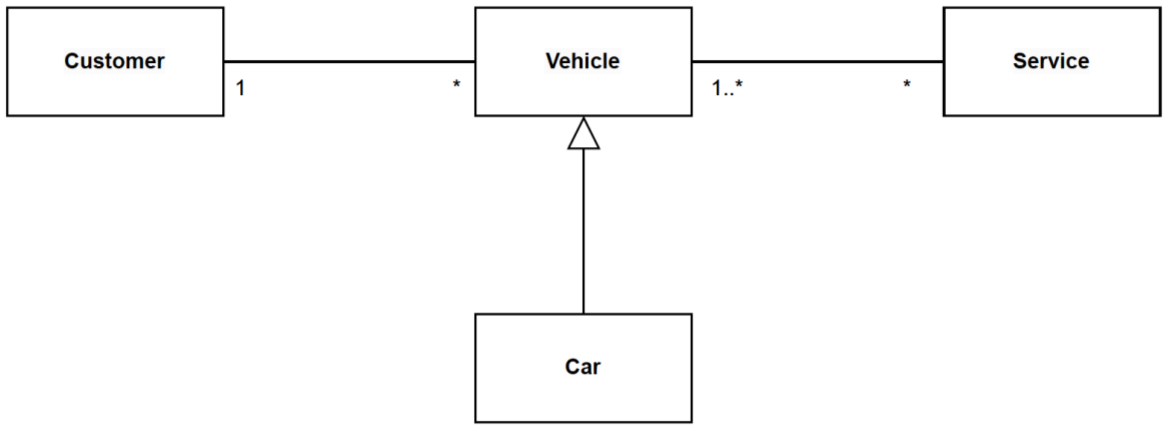
Use Case:	Generate Invoice
Notes:	Extends “Discount”
Trigger:	The customer wants to get an invoice
Main Scenario:	
1.	The customer asks the appointment and billing team to generate the invoice.
2.	The appointment and billing team asks the customer to provide the username and the pin code.
3.	The appointment and billing team asks the customer to verify the vehicle information.
4.	The appointment and billing team adds the customer and vehicle information to the invoice.
5.	The appointment and billing team asks the customer to verify the list of services that are done.
6.	The appointment and billing team adds the list of services that are done to the invoice.
7.	>>Extends Apply Discount use case<<
8.	The appointment and billing team calculates the taxes, total, discount, and final amount.
9.	The appointment and billing team prints a copy of the invoice for the customer.
Exceptions:	
2a.	1. The username and pin code does not match. 2. The appointment and billing team notifies the customer and asks to enter the right username and pin code.
5a.	1. The list is not updated with all the services done. 2. The appointment and billing team updates the list of services

Use Case:	Authenticate Customer
Trigger:	The system authenticates the customer.
Preconditions:	The customer is registered in the garage system.
Main Scenario:	
1.	The garage system asks the user to provide credentials to be authenticated.
2.	The Customer provides credentials.
3.	The garage system validates that the credentials are valid.
Exceptions:	
2a.	1. The customer provides the wrong credentials. 2. The garage system shows an error message and asks the customer to enter the correct credentials.

Use Case:	Apply Discount
Trigger:	The customer has a discount code.
Preconditions:	The customer asked the appointment and billing team to generate the invoice.
Main Scenario:	
1.	The customer asks the appointment and billing team to apply a discount.
2.	The appointment and billing team asks the customer to provide the discount code.
3.	The appointment and billing team adds the discount code to the invoice.
Exceptions:	
2a.	1. The discount code is invalid. 2. The use case ends.

UML Class Diagram and Description


Customer
- firstName: String
- lastName: String
- phoneNumber: String
- email: String
- address: String
+ getFirstName(): String
+ setFirstName(firstName : String)
+ getLastName(): String
+ setLastName(lastName : String)
+ getPhoneNumber(): String
+ setPhoneNumber(phoneNumber : String)
+ getEmail(): String
+ setEmail(email : String)
+ getAddress(): String
+ setAddress(address : String)
+ displayCustomerInfo(): String
Vehicle
- ID: String
- make: ENUM
- type: ENUM
- color: ENUM
- licensePlate: String
+ getID(): String
+ setID(ID : String):
+ getMake(): ENUM
+ setMake(make : Make)
+ getType(): ENUM
+ setType(type : Type)
+ getColor(): ENUM
+ setColor(color : Color)
+ getLicensePlate(): String
+ setLicensePlate(licensePlate: String):
+ __str__(): String
Service
- serviceName: String
- description: String
- partsCost: Float
- laborCost: Float
- totalPrice: Float
+ getServiceName(): String
+ setServiceName(serviceName : String)
+ getDescription(): String
+ setDescription(description : String)
+ getPartsCost(): Float
+ setPartsCost(partsCost : Float)
+ getLaborCost(): Float
+ setLaborCost(laborCost : Float)
+ getTotalPrice(): Float
+ setTotalPrice(totalPrice : Float)
+ displayServiceInfo(): String
Car
- seaters : ENUM
+ setSeaters(seaters: Seaters)
+ getSeaters(): ENUM
+ __str__(): String


Customer
1
*
Vehicle
1..*
*
Service
Car

Vehicle and Car

The Car class inherits from the Vehicle class, thus it includes all of its attributes (make, type, color, ID, and licensePlate), in addition to additional attributes and methods particular to cars (seaters). An arrow with a hollow triangular head pointing from Car to Vehicle indicates that the link between the two objects is an inheritance relationship.

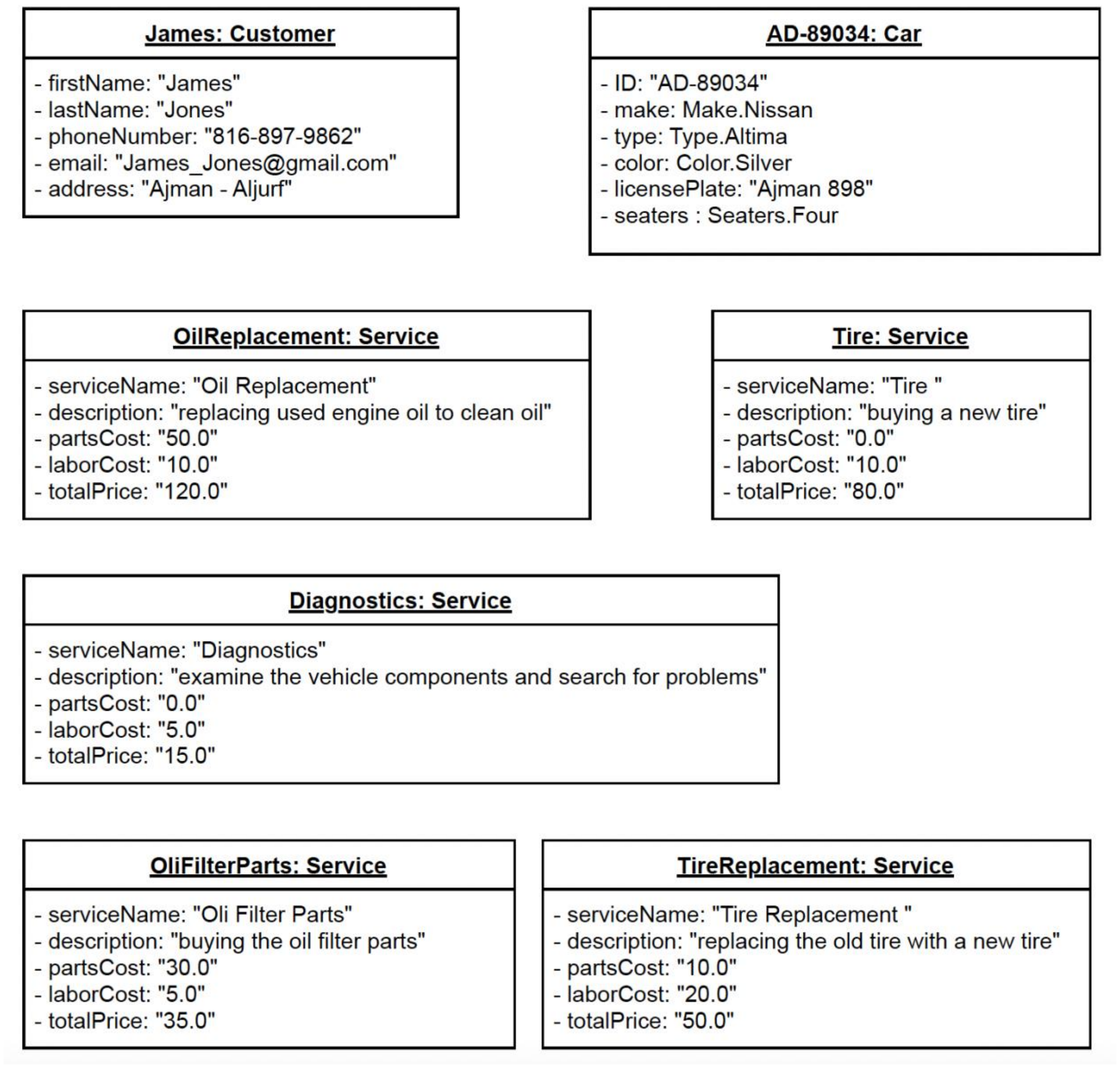
Vehicle and Customer

A Vehicle class has a binary association with the Customer class. As a result, a Customer has at least no vehicle and at most many Vehicles, but a Vehicle has one Customer only.

Vehicle and Service

A Vehicle class has a binary association with Service class. This means that while a Vehicle has at least no service and at most many Services, each Service has at least one Vehicle and at most many Vehicles.

To sum up, the UML diagram illustrates a car garage scenario in which a Customer may own one or more Vehicles, each of which may be a Car or another form of vehicle, and each of which may be accompanied by no or more Services. The UML diagram uses multiple arrows to represent the connections and interconnections between the classes.



Python code

```
#importing Enum
from enum import Enum

# Creating a Make enum class with different make types
class Make(Enum):
    Audi = 1
    Nissan = 2
    Lexus = 3
    Mercedes = 4
    Tesla = 5

# Creating a Type enum class with different types for each make
class Type(Enum):
  #Audi Type
    A2 = 1
    A3 = 2
    A4 = 3
    A5 = 4
    A6 = 5
    A7 = 6
    Q3 = 7
    Q5 = 8
    Q7 = 9

  #Nissan Type
    Sunny = 10
    Patrol = 11
    Altima = 12

  #Lexus Type
    IS = 13
    ES = 14
    GS = 15
    UX = 16
    NX = 17
    GX = 18
    LX = 19

  #Mercedes Type
    AClass = 20
    CClass = 21
    EClass = 22
    GClass = 23

  #Tesla Type
    ModelS = 24
    ModelX = 25
    ModelY = 26

# Creating a Color enum class with different color options
class Color(Enum):
    Black = 1
    White = 2
    Red = 3
    Blue = 4
    Silver = 5

# Creating a Seaters enum class with different seating options
class Seaters(Enum):
    Two = 1
    Four = 2
    Five = 3
    Seven = 4
    Eight = 5

# Creating a Customer class
class Customer:
    def __init__(self, firstName, lastName, phoneNumber, email, address):
# Initializing the attributes of the class
        self.__firstName = firstName
        self.__lastName = lastName
        self.__phoneNumber = phoneNumber
        self.__email = email
        self.__address = address

    # Defining getters and setters for each attribute
    def getFirstName(self):
        return self.__firstName

    def setFirstName(self, firstName):
        self.__firstName = firstName

    def getLastName(self):
        return self.__lastName

    def setLastName(self, lastName):
        self.__lastName = lastName

    def getPhoneNumber(self):
        return self.__phoneNumber

    def setPhoneNumber(self, phoneNumber):
        self.__phoneNumber = phoneNumber

    def getEmail(self):
        return self.__email

    def setEmail(self, email):
        self.__email = email
    def getAddress(self):
        return self.__address

    def setAddress(self, address):
        self.__address = address

    # Defining a function that displays customer information
    def displayCustomerInfo(self):
        print("Customer:", self.__firstName, ", Last Name: ",
self.__lastName, ", phoneNumber: ", self.__phoneNumber, ", Email: " ,
self.__email, ", Address:", self.__address)

# Creating a Vehicle class
class Vehicle:
    def __init__(self, ID, make, type, color, licensePlate): #
Initializing the attributes of the class
        self.__ID = ID
        self.__make = make
        self.__type = type
        self.__color = color
        self.__ID = ID
        self.__licensePlate = licensePlate

    # Defining getters and setters for each attribute
    def getID(self):
        return self.__ID

    def setID(self, ID):
        self.__ID = ID

    def getMake(self):
        return self.__make

    def setMake(self, make):
        self.__make = make

    def getType(self):
        return self.__type

    def setType(self, type):
        self.__type = type

    def getColor(self):
        return self.__color
    def setColor(self, color):
        self.__color = color

    def getLicensePlate(self):
        return self.__licensePlate

    def setLicensePlate(self, licensePlate):
        self.__licensePlate = licensePlate

    #__str__ method to return a string representation of the object
    def __str__(self):
        return "ID:" + self.__ID+ ", Make:"+ self.__make.name+ ", Type:"+
self.__type.name+ ", Color:" + self.__color.name+  ", License Plate:"+
self.__licensePlate

# Creating a Car class that inherts from Vehicle class
class Car(Vehicle):
    def __init__(self,ID, make, type, color, licensePlate, seaters): #
Initializing the attributes of the class
        super().__init__(ID, make, type, color,  licensePlate) #inherting
the attributes of the parent class using super()
        self.__seaters = seaters

    # Defining getters and setters for each attribute
    def setSeaters(self, seaters):
        self.__seaters = seaters

    def getSeaters(self):
        return self.__seaters

    #__str__ method to return a string representation of the object
    def __str__(self):
        return super().__str__() + ", Seaters: "+ self.__seaters.name
#inherting the __str__  of the parent class using super() and adding the
seater

# Creating a Service class
class Service:
    def __init__(self, serviceName, description, partsCost, laborCost,
totalPrice): # Initializing the attributes of the class
        self.__serviceName = serviceName
        self.__description = description
        self.__partsCost = partsCost
        self.__laborCost = laborCost
        self.__totalPrice = totalPrice

    # Defining getters and setters for each attribute
    def getServiceName(self):
        return self.__serviceName

    def setServiceName(self, serviceName):
        self.__serviceName = serviceName

    def getDescription(self):
        return self.__description

    def setDescription(self, description):
        self.__description = description

    def getPartsCost(self):
        return self.__partsCost

    def setPartsCost(self, partsCost):
        self.__partsCost = partsCost

    def getLaborCost(self):
        return self.__laborCost

    def setLaborCost(self, laborCost):
        self.__laborCost = laborCost

    def getTotalCost(self):
        return self.__totalCost

    def setTotalCost(self, totalCost):
        self.__totalCost = totalCost

    # Defining a function that displays service information
    def displayServiceInfo(self):
        print("Service Name:", self.__serviceName, ", Description:",
self.__description, ", Parts Cost:" , self.__partsCost, ", Labor Cost:",
self.__laborCost, ", Total Price:", self.__totalPrice)


# Creating objects and testing the functions

Customer1 = Customer("James", "Jones", "816-897-9862",
"James_Jones@gmail.com", "Ajman - Aljurf")
Customer1.displayCustomerInfo()
```

```
Vehicle1 = Vehicle(ID = "AD-89034", make = Make.Nissan, type =
Type.Altima, color = Color.Silver, licensePlate= "Ajman 898")
print(Vehicle1)
Vehicle1.setColor(Color.Black)
print(Vehicle1)


Car1 = Car(ID = "AD-89034", make = Make.Nissan, type = Type.Altima, color
= Color.Silver, licensePlate= "Ajman 898", seaters = Seaters.Four)
print(Car1)
Car1.getSeaters()


Diagnostics = Service(serviceName = "Diagnostics" , description = "examine
the vehicle components and search for problems" , partsCost = "0.0" ,
laborCost = "5.0", totalPrice = "15.0" )
Diagnostics.displayServiceInfo()


OilReplacement = Service(serviceName = "Oil Replacement" , description =
"replacing used engine oil to clean oil" , partsCost = "35.0" , laborCost
= "10.0", totalPrice = "120.0" )
OilReplacement.displayServiceInfo()

OliFilterParts = Service(serviceName = "Oli Filter Parts" , description =
"buying the oil filter parts" , partsCost = "30.0" , laborCost = "5.0",
totalPrice = "35.0" )
OliFilterParts.displayServiceInfo()

TireReplacement = Service(serviceName = "Tire Replacement" , description =
"replacing the old tire with a new tire" , partsCost = "10.0" , laborCost
= "20.0", totalPrice = "50.0" )
TireReplacement.displayServiceInfo()

Tire = Service(serviceName = "Tire" , description = "buying a new tire" ,
partsCost = "0.0" , laborCost = "10.0", totalPrice = "80.0" )
Tire.displayServiceInfo()
```

GitHub Repository Link

https://github.com/MaryamAlwars/Assignment-1-Software-Modelling-

Summary of learnings

I was able to use this assignment to put my understanding of a variety of important software development concepts to use in a car garage system. In particular, I was competent to

- Build a UML class diagram: I made a UML class diagram that correctly depicted the connections between the Customer, Vehicle, Car, and Service classes in the car garage system.
- Create a use case diagram: In order to identify the many use cases in the system, such as creating an account, scheduling appointments, repairing vehicles, and generating invoices, I constructed a use case diagram successfully.
- Use inheritance and association: In order to build the Car class, which inherits from the Vehicle class, i used inheritance. Also, I showed the relationships among the different classes in the system using associations.
- Develop Python code: I developed Python code for the classes. Each includes its attributes and functions of it. Also, I created an object for each class and tested the code. I was able to use what I learned about super() to inherit the attributes of the parent class.

I was able to accomplish this in order to obtain a deeper comprehension of how these ideas can be used to design and develop software applications that cater to the requirements of users in a number of various scenarios. This shows that I was able to effectively apply my understanding of use case diagrams, UML class diagrams, Python programming, and inheritance and association in a practical situation. I was able to comprehend these ideas and how they could be used in the software development process more fully as a result of accomplishing this assignment. Also, I used GitHub to show my progress and how I improve and modify my work. This degree of satisfaction with my work is a sign that I am improving as a software developer, and it also shows how beneficial the learning process has been for me.