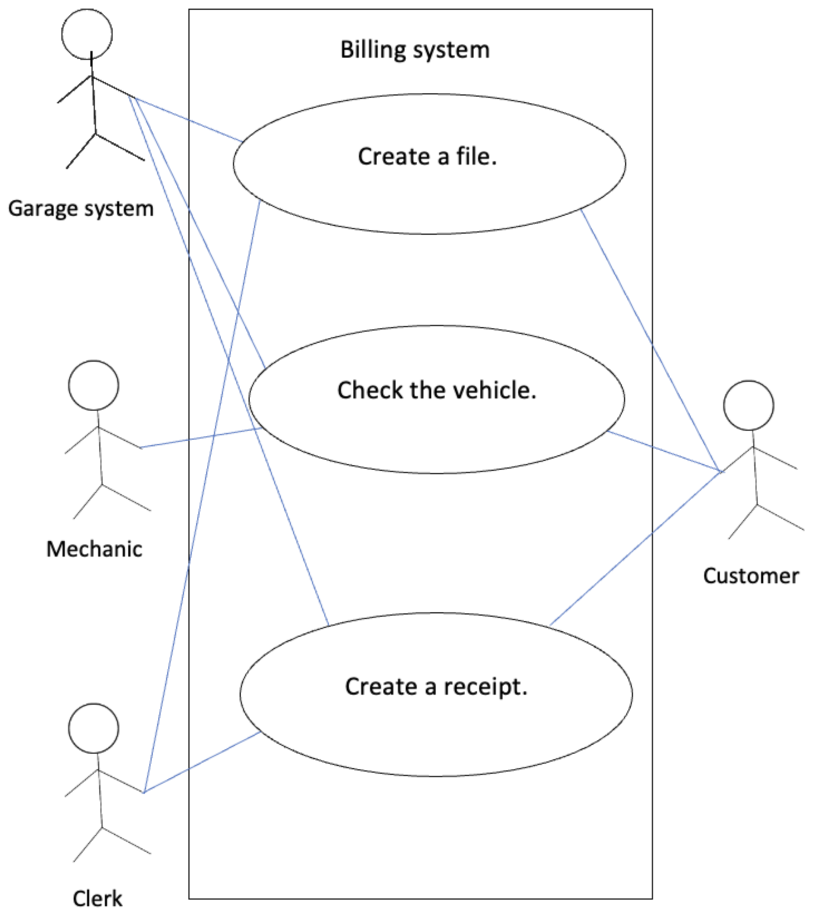
1- **UML Use-Case **Diagram and Description.



First, creating a file to the customer will help with another services and ensure the bill is created effectively, and efficiently. It is a use case that relates to the figure provided since it will store the information and ease the process.

Use case:	Create a file
Trigger:	The customer wants to have a file to store his information before doing the services on the garage.
Precondition:	The customer is authenticated.
Main scenarios:	
1-	The customer reaches out the clerk.
2-	The clerk provides the customer with a list to fill in.
3-	The customer specifies their information.
4-	The clerk enters the information into the system.
5-	The system stores the information provided.
6-	The system displays the information.

7-	The clerk singles to the system that everything is correct to start creating a file
8-	The system creates a file with the information provided.
Exceptions:	
7a	1- The information is wrong or missing 2- The system notifies the cleck to adjust it with the customer.

Second, checking the vehicle is an important use case to include since it shows the process the customer will go through. It relates to the figure as it shows the services the customer had at the garage and having that information will guide us with creating the bill later.

Use case:	Check the vehicle
Trigger:	The customer needs his vehicle to be checked on the garage.
Precondition:	The customer is authenticated
Main scenarios:	
1-	The customer enters their personal details into the system.
2-	The customer enters the services they want.
3-	The system stores the information provided.
4-	The system checks if the services are available.
5-	The system checks who is the responsible mechanic for those services.
6-	The system notifies the mechanic that they have a customer.
7-	The mechanic does the services.
Exceptions:	
4a	1-The services are not available. 2-The customer chooses a different service.
5a	1- The mechanic is not available. 2- The system looks for a substitute.

Thirdly, the receipt or invoice will be created which is the most important use case that will help the car garage manage their billing system effectively. It ensures the information, and calculations are provided, and created appropriately.

Use Case:	Create a receipt
Trigger:	The car garage wants to create a receipt to manage bills.

Precondition:	The customer is authenticated
Main scenarios:	
1-	The customer enters their personal details.
2-	The clerk receives the customer’s information.
3-	The clerk check who was the mechanic.
4-	The clerk enters the mechanic’s name and date of the services.
5-	The clerk ensures all the information are entered and appropriate.
6-	The system stores the information.
7-	The clerk enters the customer chosen services into the system.
8-	The system displays the amount of each service.
9-	The system calculates the total amount and taxes.
10-	The system calculates discounts.
11-	The system displays the final number of services.
12-	The system creates the bill with all the information.
13-	The clerk prints the bill and upload it to the system.
Exceptions:	
10a	The customer might not have a discount.

2- **UML Class **Diagram and Description.

List of Classes and attributes:

-Customer: firstName, lastName, cellPhoneNumber, gender, dateOfBirth.

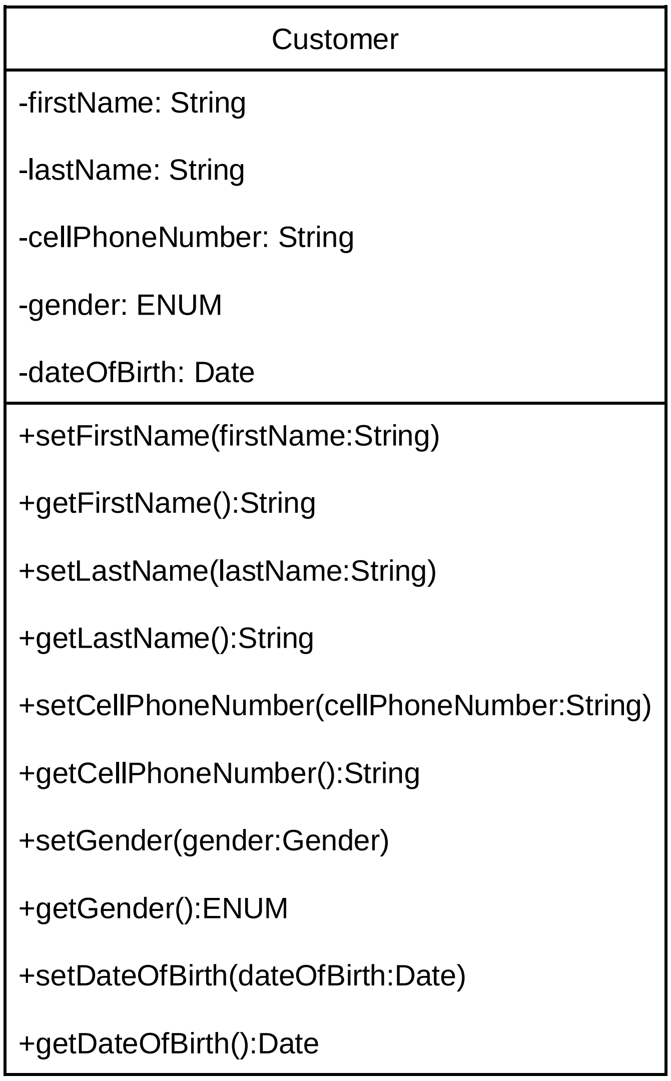
-Service: serviceName, servicePrice, mechanicName, date, numberOfServices.

-Vehicle: type, color, id, yearOfVehicle, make.

-Car: doors, type, color, id, yearOfVehicle, make.

-Price: taxes, total, discount, finalAmount, servicePrice.

The class customer has 5 attributes with their type and their setter and getter function.

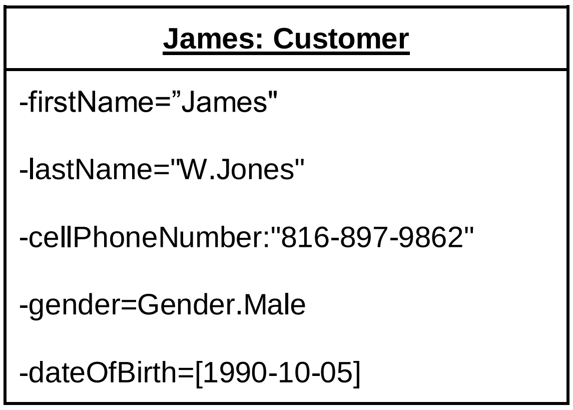


Customer

-firstName: String
-lastName: String
-cellPhoneNumber: String
-gender: ENUM
-dateOfBirth: Date

+setFirstName(firstName:String)
+getFirstName():String
+setLastName(lastName:String)
+getLastName():String
+setCellPhoneNumber(cellPhoneNumber:String)
+getCellPhoneNumber():String
+setGender(gender:Gender)
+getGender():ENUM
+setDateOfBirth(dateOfBirth:Date)
+getDateOfBirth():Date

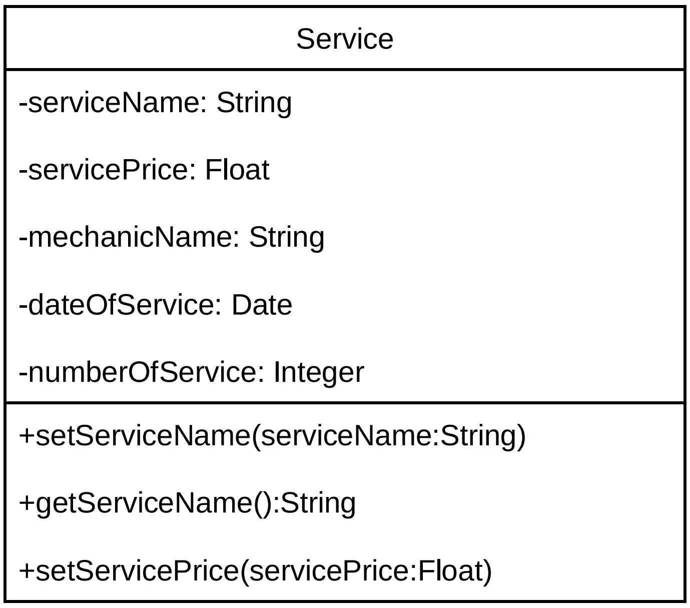
Object1



<u>James: Customer</u>

-firstName=”James"
-lastName="W.Jones"
-cellPhoneNumber:"816-897-9862"
-gender=Gender.Male
-dateOfBirth=[1990-10-05]

The class Service has 5 attributes with their type and their setter and getter function are included.



Service

-serviceName: String
-servicePrice: Float
-mechanicName: String
-dateOfService: Date
-numberOfService: Integer

+setServiceName(serviceName:String)
+getServiceName():String
+setServicePrice(servicePrice:Float)

+getServicePrice():Float

+setMechanicName(mechanicName:String)

+getMechanicName():String

+setDateOfService(dateOfService:Date)

+getDateOfService():Data

+setNumberOfService(numberOfService:Integer)

+getNumberOfService():Integer

Object 1

<u>Service1: Service</u>

-serviceName="Diagnostics"

-servicePrice=”15”

-mechanicName=”Hans K"

-dateOfService="March 13, 2022"

-numberOfService=”1”

Object 2

<u>Service2: Service</u>

-serviceName="Oil Replacement"

-servicePrice=”120”

-mechanicName=”Hans K"

-dateOfService="March 13, 2022"

-numberOfService=”2”

Object 3

<u>Service3: Service</u>

-serviceName="Oil Filter Parts"

-servicePrice=”35”

-mechanicName=”Hans K"

-dateOfService="March 13, 2022"

-numberOfService=”3”

Object 4

<u>Service4: Service</u>

-serviceName="Tire Replacement (2)"

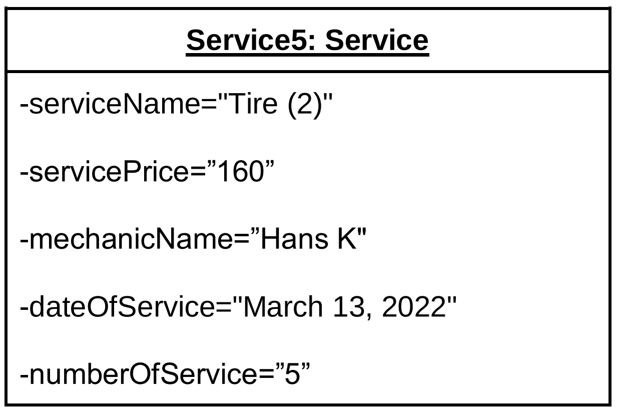
-servicePrice=”100”

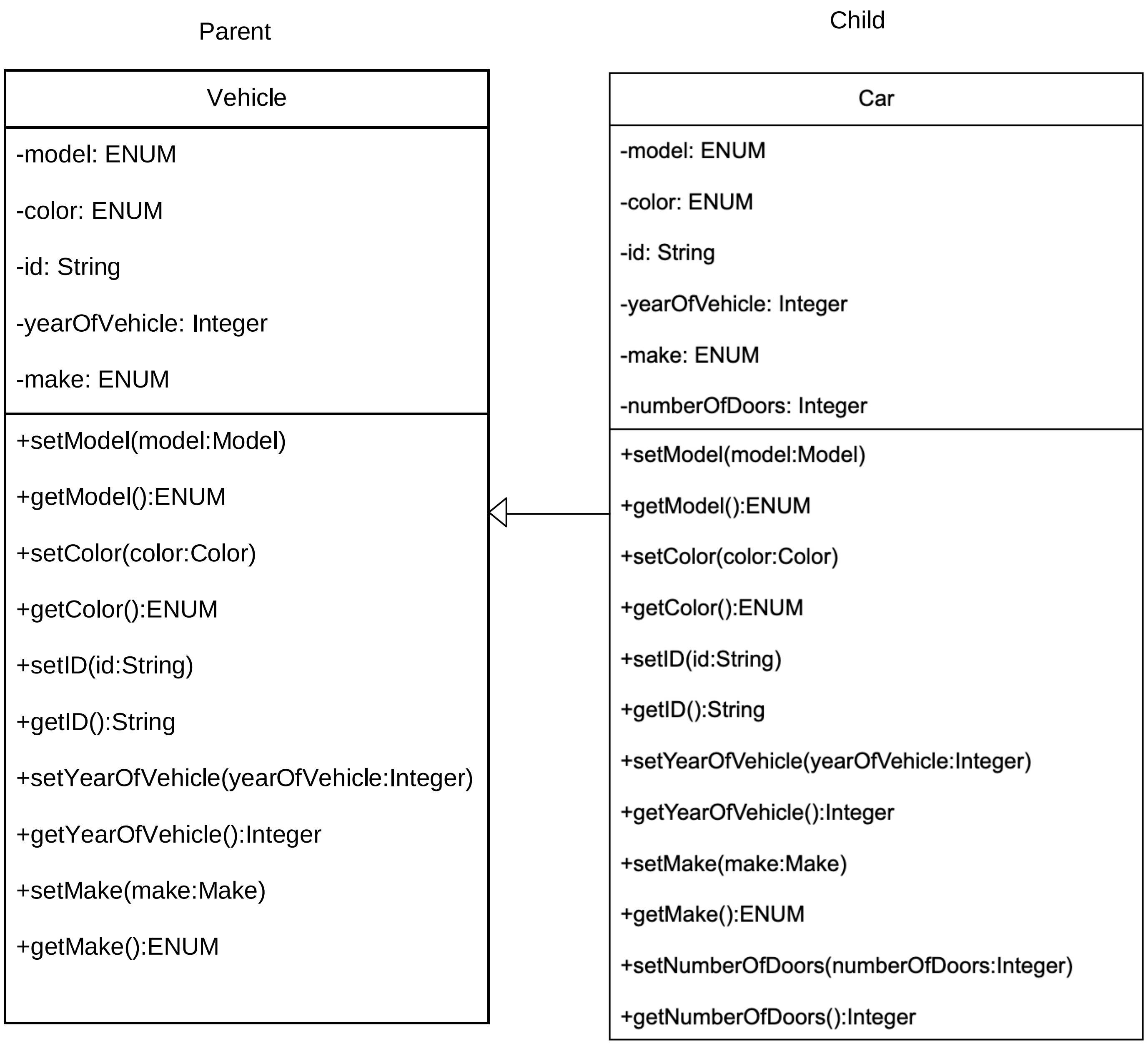
-mechanicName=”Hans K"

-dateOfService="March 13, 2022"

-numberOfService=”4”

Object 5

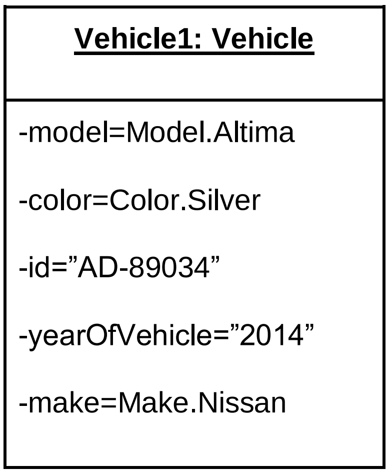




The class Car has 6 attributes with their type and their setter and getter function are included. It is the child class of the class Vehicle, and it inherits its attributes. One attribute “numberOfDoors” is related to Cars specifically since the vehicle can be motorcycles, boats, or airplanes, where the concept of doors may not apply. There is a “is a” relationship between them where “A car is a Vehicle”, so car is as subclass of Vehicle, and it is a Single inheritance. We can refer to the base case on the code using the super class or just by calling the class itself again on the subclass. The class Vehicle has 5 attributes with their type and their setter and getter function are included. It is the parent class of the class car.

Objects of class Vehicle and Car

Object 1 **Object 1**



Vehicle1: Vehicle

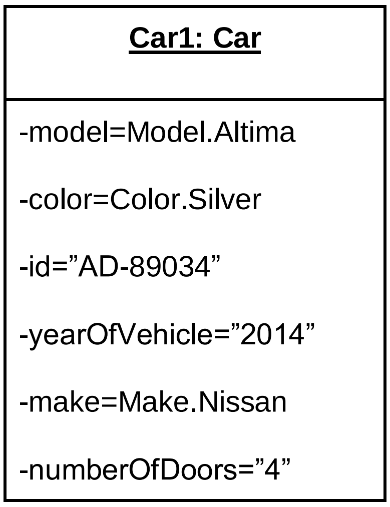
-model=Model.Altima

-color=Color.Silver

-id=”AD-89034”

-yearOfVehicle=”2014”

-make=Make.Nissan



Car1: Car

-model=Model.Altima

-color=Color.Silver

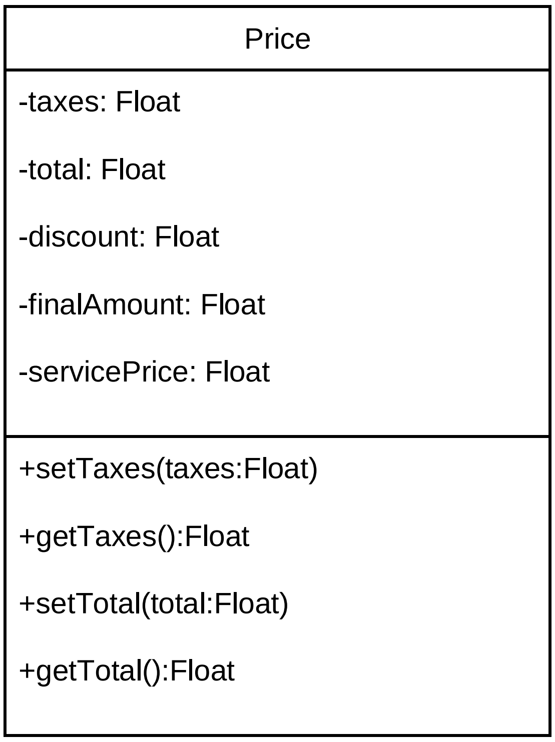
-id=”AD-89034”

-yearOfVehicle=”2014”

-make=Make.Nissan

-numberOfDoors=”4”

The class Price has 5 attributes with their type and their setter and getter function are included.



Price

-taxes: Float

-total: Float

-discount: Float

-finalAmount: Float

-servicePrice: Float

+setTaxes(taxes:Float)

+getTaxes():Float

+setTotal(total:Float)

+getTotal():Float

+setDiscount(discount:Float)

+getDiscount():Float

+setFinalAmount(finalAmount:Float)

+getFinalAmount():Float

+setServicePrice(servicePrice:Float)

+getServicePrice():Float

The other relationships between those classes might be as an association, but not inheritance, the only one is the vehicle and the car.

Object 1

<u>Price1: Price</u>
-taxes=”21.5”
-total=”451.5”
-discount=”11.5”
-finalAmount=”440”
-servicePrice=15”

Object 2

<u>Price2: Price</u>
-taxes=”21.5”
-total=”451.5”
-discount=”11.5”
-finalAmount=”440”
-servicePrice=”120”

Object 3

<u>Price3: Price</u>
-taxes=”21.5”
-total=”451.5”
-discount=”11.5”
-finalAmount=”440”
-servicePrice=”35”

Object 4

Price4: Price
-taxes=”21.5”
-total=”451.5”
-discount=”11.5”
-finalAmount=”440”
-servicePrice=”100”

Object 5

Price5: Price
-taxes=”21.5”
-total=”451.5”
-discount=”11.5”
-finalAmount=”440”
-servicePrice=”160”

3- ****Python classes **(copy paste the code, NOT an image of the code) i. The code must be well documented with good coding standards followed.**

```
#First, we imported the enumerator type to limit the user choices and
make the code more robust'
from enum import Enum

class Gender(Enum):
    Female=1
    Male=2
#We create a classs called customer and define the instructor to
initializes the object with the provided attributes, and we define each
attribute, then we have the setter functions
#to set the values of the object attributes and getter functions to get
the values of the objects attributes. We have the'__' before each
attribute to ensure they are private'
class Customer:
    def __init__(self, firstName, lastName, cellPhoneNumber, gender,
dateOfBirth):
        self.__firstName = firstName
        self.__lastName = lastName
        self.__cellPhoneNumber = cellPhoneNumber
        self.__gender = gender
        self.__dateOfBirth = dateOfBirth

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

    def getFirstName(self):
        return self.__firstName

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

    def getLastName(self):
        return self.__lastName

    def set_CellPhoneNumber(self, cellPhoneNumber):
        self.__cellPhoneNumber = cellPhoneNumber

    def getCellPhoneNumber(self):
        return self.__cellPhoneNumber

    def set_gender(self, gender):
        self.__gender = gender

    def get_gender(self):
        return self.__gender

    def setDateOfBirth(self, dateOfBirth):
        self.__dateOfBirth = dateOfBirth

    def getDateOfBirth(self):
        return self.__date_of_birth
#we define a function that will help us get all the information at once
which much more efficient than getting them one by one.'
    def printInfo(self):
        print("First Name: ",self.__firstName,", Last Name:
",self.__lastName," ","Cell Phone Number:"," ",self.__cellPhoneNumber,
" ", "Gender:", self.__gender.name," ","Date Of
Birth:",self.__dateOfBirth)
#we create an object to show actual values of the attributes of the
class and print the info of it
Customer1=Customer("James","W.Jones","816-897-9862",Gender.Male,
"[1990-10-05]")
Customer1.printInfo()
```

```
#We create a classs called Service and define the instructor to
initializes the object with the provided attributes, and we define each
attribute, then we have the setter functions
```

```python
#to set the values of the object attributes and getter functions to get
the values of the objects attributes. We have the'__' before each
attribute to ensure they are private'
class Service:
    def __init__(self, serviceName, servicePrice, mechanicName,
dateOfService, numberOfService):
        self.__serviceName = serviceName
        self.__servicePrice = servicePrice
        self.__mechanicName = mechanicName
        self.__dateOfService = dateOfService
        self.__numberOfService = numberOfService

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

    def getServiceName(self):
        return self.__serviceName

    def setServicePrice(self, servicePrice):
        self.__servicePrice = servicePrice

    def getServicePrice(self):
        return self.__servicePrice

    def setMechanicName(self, mechanicName):
        self.__mechanicName = mechanicName

    def getMechanicName(self):
        return self.__mechanicName

    def setDateOfService(self, dateOfService):
        self.__dateOfService = dateOfService

    def getDateOfService(self):
        return self.__dateOfService

    def setNumberOfService(self, numberOfService):
        self.__numberOfService = numberOfService

    def getNumberOfService(self):
        return self.__numberOfService
 #we define a function that will help us get all the information at
once which much more efficient than getting them one by one.'
    def printInfo(self):
        print("Service Name:",self.__serviceName,", Service Price:
",self.__servicePrice,", Mechanic Name:",self.__mechanicName, ", Date
Of Service:", self.__dateOfService,", Number Of
Service:",self.__numberOfService)
```

```
#we create an object for each service to show actual values of the
attributes of the class and print the info of it
Service1=Service("Diagnostics","15","Hans K","March 13, 2022","1")
Service2=Service("Oil Replacement","120","Hans K","March 13, 2022","2")
Service3=Service("Oil Filter Parts","35","Hans K","March 13, 2022","3")
Service4=Service("Tire Replacement (2)","100","Hans K","March 13,
2022","4")
Service5=Service("Tire (2)","160","Hans K","March 13, 2022","5")
Service1.printInfo()
print("-------")
Service2.printInfo()
print("-------")
Service3.printInfo()
print("-------")
Service4.printInfo()
print("-------")
Service5.printInfo()
```

```
#We create a classs called Price and define the instructor to
initializes the object with the provided attributes, and we define each
attribute, then we have the setter functions
#to set the values of the object attributes and getter functions to get
the values of the objects attributes. We have the'__' before each
attribute to ensure they are private'
class Price():
    def __init__(self, servicePrice, total, taxes, discount,
finalAmount):
        self.__servicePrice= servicePrice
        self.__total = total
        self.__taxes = taxes
        self.__discount = discount
        self.__finalAmount = finalAmount

    def setServicePrice(self, servicePrice):
        self.__servicePrice = servicePrice

    def getServicePrice(self):
        return self.servicePrice

    def setTotal(self, total):
        self.__total = total

    def getTotal(self):
        return self.__total

    def setTaxes(self, taxes):
        self.__taxes = taxes
    def getTaxes(self):
        return self.__taxes

    def setDiscount(self, discount):
        self.__discount = discount

    def getDiscount(self):
        return self.__discount

    def setFinalAmount(self, finalAmount):
        self.__finalAmount = finalAmount

    def getFinalAmount(self):
        return self.__finalAmount
#we define a function that will help us get all the information at once
which much more efficient than getting them one by one.'
    def printInfo(self):
        print(" Service Price: ",self.__servicePrice,", Total:
",self.__total, ",Taxes:", self.__taxes,",
Discount:",self.__discount,", Final Amount:",self.__finalAmount)
#we create an object for the price of each service to show actual
values of the attributes of the class and print the info of it
Price1=Price("15","451.5","21.5", "11.5", "440")
Price2=Price("120","451.5","21.5", "11.5", "440")
Price3=Price("35","451.5","21.5", "11.5", "440")
Price4=Price("100","451.5","21.5", "11.5", "440")
Price5=Price("160","451.5","21.5", "11.5", "440")
Price1.printInfo()
print("-------")
Price2.printInfo()
print("-------")
Price3.printInfo()
print("-------")
Price4.printInfo()
print("-------")
Price5.printInfo()
```

```
#First, we imported the enumerator type to limit the user choices and
make the code more robust'
from enum import Enum

class Make(Enum):
    Toyota=1
    VolksWagen=2
    Nissan=3

class Model(Enum):
    Yaris=1
    Altima=2
    Pasat=3

class Color(Enum):
    White=1
    Silver=2
    Black=3
#We create a classs called Vehicle and define the instructor to
initializes the object with the provided attributes, and we define each
attribute, then we have the setter functions
#to set the values of the object attributes and getter functions to get
the values of the objects attributes. We have the'__' before each
attribute to ensure they are private'
class Vehicle:#Parent class
    def __init__(self, make, model, yearOfVehicle, color, id):
        self.__make = make
        self.__model = model
        self.__yearOfVehicle = yearOfVehicle
        self.__color = color
        self.__id = id
    def setMake(self, make):
        self.__make = make

    def getMake(self):
        return self.__make

    def setModel(self, model):
        self.__model = model

    def getModel(self):
        return self.__model

    def setYearOfVehicle(self, yearOfVehicle):
        self.__yearOfVehicle = yearOfVehicle

    def getYearOfVehicle(self):
        return self.__yearOfVehicle

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

    def getColor(self):
        return self.__color

    def setID(self, id):
        self.__id = id

    def getID(self):
        return self.__id
#we define a function that will help us get all the information at once
which much more efficient than getting them one by one.'
    def displayInfo(self):
        print('Make =',self.__make.name,', Model=',self.__model.name,',
yearOfVehicle=',self.__yearOfVehicle,', Color=', self.__color.name,',
ID=',self.__id)

class Car(Vehicle):#Child class
    def __init__(self, make, model, yearOfVehicle, color, id,
numberOfDoors):
        Vehicle.__init__(self, make, model, yearOfVehicle, color,
id)#refer to the parent class
        #number of doors is specfic attribute for the car since the
vehicle can be motorcycles, boats, or airplanes, where the concept of
doors may not apply.
        self.__numberOfDoors = numberOfDoors

    def displayInfo(self):
      Vehicle.displayInfo(self)
      print('numberOfDoors=', self.__numberOfDoors)
#we create an object for car and Vehicle to show actual values of the
attributes of the class and print the info of it
Car1=Car(Make.Nissan,Model.Altima, "2014", Color.Silver, "AD-89034",
"4")
Vehicle1=Vehicle(Make.Nissan,Model.Altima, "2014", Color.Silver, "AD-
89034")

Vehicle1.displayInfo()
print("-------")
Car1.displayInfo()
```

4- **GitHub link:** https://github.com/202101145/Assignment1-ICS-220

5- **Summary of learning.**

Referring to our Los in the assignment, I had the opportunity to apply what I learned about Object Oriented programming and UML notations. We were asked to create and design a software that map real-world entities and here it was the billing management system for a car garage. Billing management is an important process that must be done carefully, and through this assignment I believe the use cases I created was effective to do so.
The diagrams helped with representing the relationships between actors and the descriptions table showed the details of the use cases. The classes and objects were related to the system and solved the problem following a python code. The code was structured, error free, and cleared the issues. The usage of inheritance was effective as well since I did not have to repeat the code and saved space.
Now, I understand and appreciate having these programs and languages that can ease many processes and help us solve computational and real-world problems.