ICS220 Program. Fund.

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

Professor Kuhail

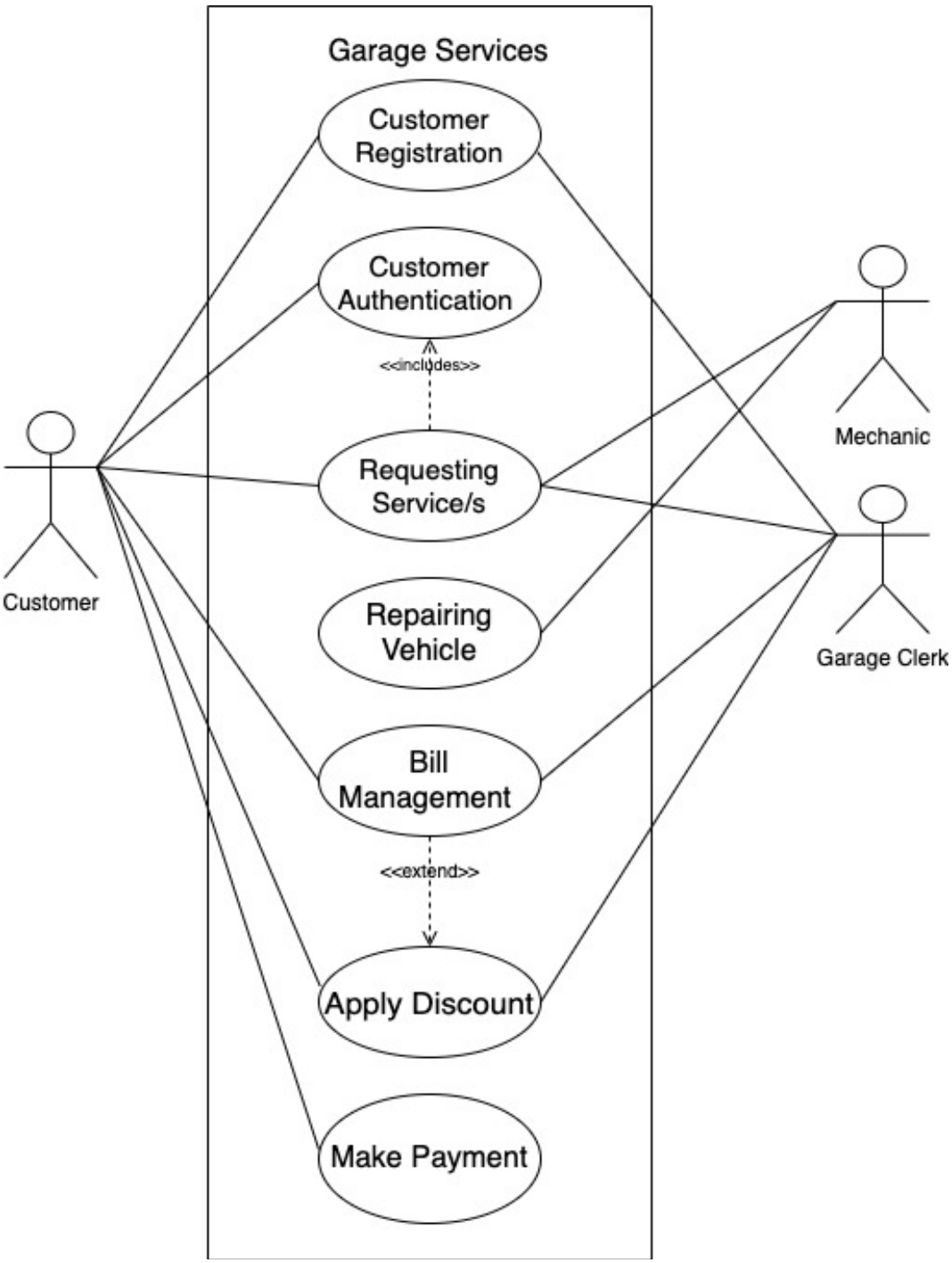
Aisha Lihwaidi

202110522

CIS

UML Use-Case Diagram

Customer: This is the person that initiates the system
Mechanic: This is the person that will take care of fixing/repairing a customers' car. Therefore, they interact with the use case "request service" to know what services the user requested, and they interact with the use case "repair vehicle" to know what request they must process and how many vehicles they must repair.



UML Use-Case Descriptions:

Upon entering the garage, the garage clerks register the customer into the system. Once the customer is registered with their personal details and vehicle details, they can be authenticated and proceed to request a service via the system. Once the customer requests a service, they are given a request ID which can be used by the customer, the garage clerk, and the mechanic for reference. The next step involves the mechanic signaling to the system they are ready to repair a vehicle and they get all the requests ID. Therefore, the mechanic can start repairing vehicles as per the customer's request. Next, the customer asks the garage clerk for their bill and their total price is calculated and displayed in the form of a receipt. Finally, the customer pays for the services

Customer Registration: This use case represents the start of the customers journey when it comes to repairing their vehicle. The way this use case works is that a garage clerk registers the customer via their cell phone number. When the customer is being registered, they provide their personal details and vehicle details to be stored for future reference.
Customer Authentication: The customers credentials are authenticated before they can use the system
Request Services: This use case represents the steps taken when the customer decides that they want to request specific services to be done on their vehicle. In this use case the customer is also provided with a request ID which can also be accessed by the mechanic and garage clerk for references.
Repairing Vehicle: In this use case the mechanic accesses the request of users to see what work he/she has ahead of him/her. They are also able to update the status of customers' requests in this use case
Bill Management: This use case is triggered when the customer asks for their receipt. The garage clerk searches the system for the customers information, their vehicle information, the mechanic information, the requested services, and calculating the discounts if applicable
Apply discount: This use case is extended in the case that the customer wants a discount and has a loyalty card
Make payment: This use case involves the customer paying the final amount for their chosen services.

Use Case	Customer registration
Trigger	The customer wants to be registered into the system
Main Scenario:	
1.	The customer signals to the garage clerk that they want to be registered into the system
2.	The garage clerk asks the customer for their cell phone number

3.	The customers give the garage clerk their cell phone number
4.	The garage clerk asks the customer their personal details
5.	The customer gives the garage clerk their personal details
6.	The garage clerk enters the customers personal details into the system
7.	The garage clerk asks the customers for their vehicle details
8.	The customers give the garage clerk their vehicle details
9.	The garage clerk enters the vehicle details into the system
10.	The garage clerk asks the customer to verify the details
11.	The customer verifies the details
12.	The garage clerk saves the customers registration
13.	The garage clerk allows the customer to enter their password
14.	The customer enters their password
15.	The garage clerk saves the customers registration
16.	The customer is successfully registered
Exceptions:	
10a.	1. There is a mistake in the details entered by the garage clerk 2. The customer asks the garage clerk to fix the mistake

Use Case	Authenticate Customer
Trigger	The system needs to authenticate the customer
Precondition:	The customer is registered with the garage clerk
Main Scenario:	
1.	The systems ask the customer to enter their credentials to be authenticated
2.	The user enters their credentials
3.	The system verifies the entered credentials
Exceptions:	
2a.	1. The user enters the wrong credentials 2. The system displays an error message 3. The user is asked to re-enter their credentials

Use Case	Requesting Services
Trigger	The customer wants to request a service from the garage
Main Scenario:	
1.	<<Invoke Authenticate Customer use case>>
2.	The customer signals to the system that they want to request a service from the garage
3.	The system searches for the customers vehicle information
4.	The system displays customers vehicle information
5.	The customer verifies their vehicle information
6.	The system asks the user to pick from the available services
7.	The customer picks from the available services
8.	The system asks the user to confirm their chosen service/s
9.	The system asks the user to select the date and time for the service/s
10.	The user selects the date and time that is convenient for them
11.	The system verifies the chosen date and time
12.	The system finalizes the service request produces a request confirmation and a request ID for the customer
Exceptions:	
7a.	1. The desired service is not available 2. Use case is terminated
9a.	1. The selected date/time is unavailable 2. The user is projected with an error message 3. The user chooses a different date/time else the use case is terminated

Use Case	Repair Vehicle
Trigger:	The mechanic is ready to repair the vehicles
Main Scenario:	
1.	Mechanic signals to the system that they are ready to repair a vehicle
2.	The system finds and displays the request IDs

3.	The mechanic approves the request ID
4.	The mechanic repairs the vehicle
5.	The mechanic updates the request status to “Vehicle Repaired”
Exceptions:	
2a.	1. There are no current requests 2. The use case terminates

Use Case	Billing Management
Trigger	The customer wants a receipt of their service
Pre-condition	
Main Scenario:	
1.	The customer signals to the garage clerk that they want the bill to the services
2.	The customers provide the garage clerk with their cell phone number
3.	The garage clerk searches for the customer person details and vehicle details through their cell phone number
4.	The garage clerk enters the details into the receipt
5.	The garage clerk enters today’s date into the receipt
6.	The garage clerk asks the customers for the request ID
7.	The customer gives the garage clerk the request ID
8.	The garage clerk enters the request ID into the system to find the users request
9.	The garage clerk asks the customer to confirm that these are his/her requested service/s
10.	The customer confirms his/her requested service/s
11.	The system adds up the services requested by the user with their respective prices
12.	The system calculates the tax
13.	>>Extends Apply Discount use case <<
14.	The system calculates the final price of the respective services requested with the tax and discounted amount if extended
15.	The system displays the final receipt
Exceptions:	
8a.	1. The garage clerk enters the wrong request ID 2. The garage clerk asks the user for their request ID again 3. The garage clerk re-enters the request ID
10a.	1. In the case that these are not the customers requested services

	2. The garage clerk asks the customer to provide their request ID 3. The garage clerk checks the system via the request ID what services the customer requested

Use Case	Apply Discount
Trigger	The customer wants to add a discount
Pre-condition	The customer must have requested a receipt for their chosen service/s
Main Scenario:	
1.	The garage clerk asks the customer to provide their loyalty card
2.	The customer provides their loyalty card
3.	The garage clerk verifies the loyalty card
4.	The garage clerk adds discounted amount into the receipt
Exceptions:	
2a.	1. The loyalty card provided by the user is invalid or expired 2. The system produces an error message 3. The garage clerk asks the user to re-enter the loyalty card details else, the use case is terminated

Use Case	Make Payment
Trigger	The customer wants to pay for their services
Pre-condition	The customer must have requested a receipt for their chosen service/s
Main Scenario:	
1.	The customer signals to the garage clerk that they are ready to pay
2.	The garage clerk presents the customer with the available payment methods
3.	The customer chooses their desired payment method
4.	The customer pays the final amount
5.	The garage clerk confirms the payment
Exceptions:	
4a. 4b.	1. Customer chooses to pay by card and enters the invalid pin code/ card details 2. The system rejects the payment and asks the customer to try again 1. Customer pays by cash and the garage clerk does not have enough change 2. The customer must choose another payment method

Classes:

I have identified these 6 classes that will be necessary to print the data/information on the bill.

These 6 classes are:

- Person, Customer, Mechanic, Vehicle, Car, and Service

Relationships Between Classes

Customer and Mechanic inherits from Person and Car inherits from Vehicle:
Inheritance occurs between the class person and the classes customer and mechanic. Mechanic is a Person and Customer is a Person. Both mechanic and customer classes inherit all the attributes of person and have their own attributes.

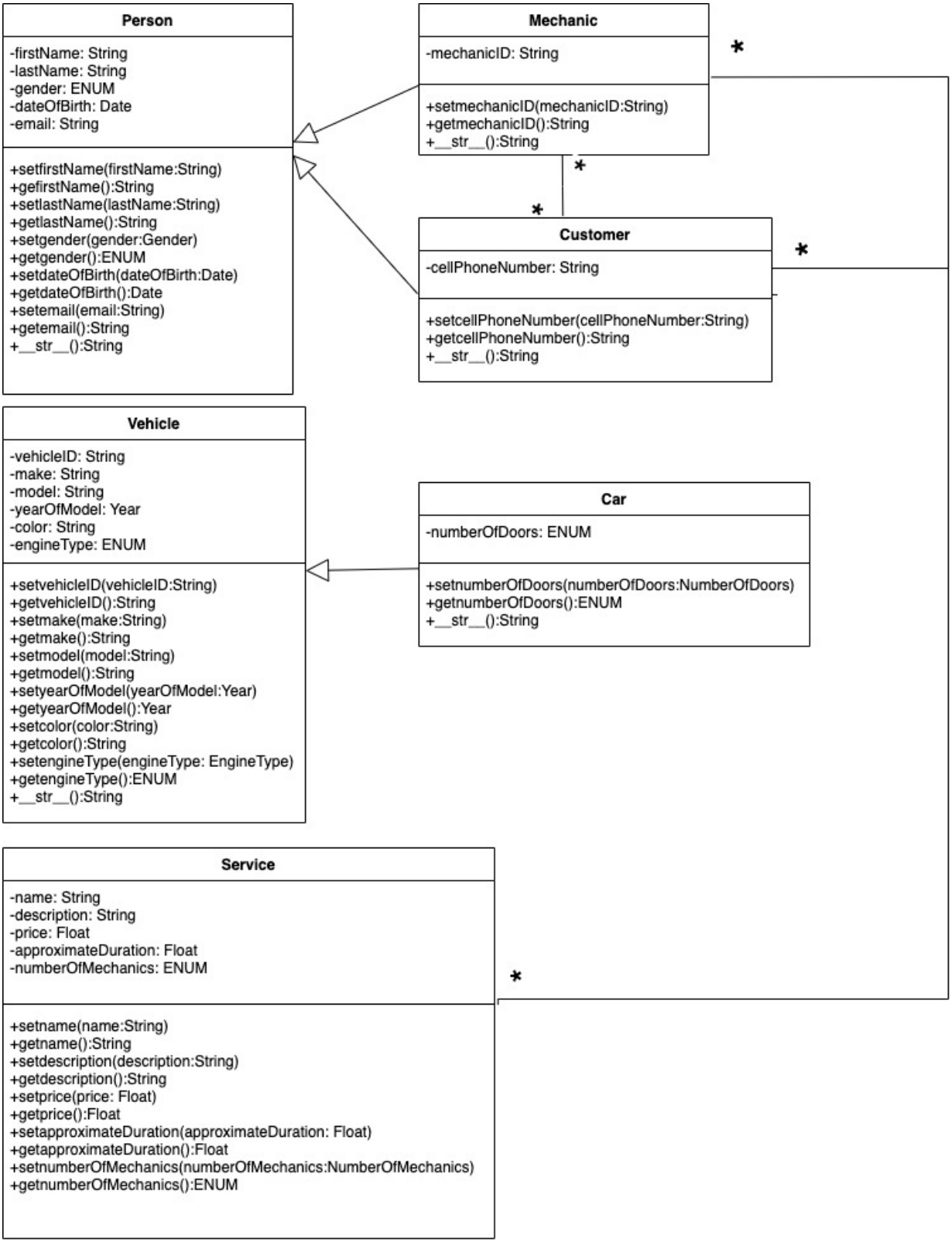
Moreover, the vehicle class and car class where the vehicle is the parent class and car are the child class. Car is a Vehicle. Car inherits all the attributes from vehicle and has its own attributes too!

Relationship between Customer and Mechanic:
Moreover, we can see a binary association between customer and mechanic. A customer can have no mechanic, or many mechanics helping repair their vehicle. They can have no mechanic if their request is yet to be processed and they can have multiple mechanics repairing their vehicle if the job needs several mechanics. Similarly, a mechanic can have no customers or multiple customers. They can have no customers if it is a slow day with not many customers coming into the garage and they can have multiple customers if it's a busy day with many customers coming in to repair their vehicles. Therefore, we can see a many to many binary associations between these two classes.

Relationship between Customer and Service:
Additionally, we can see a binary association between service and customer. A customer can request multiple services and in the same way a service can be requested by multiple customers. Therefore, we can see a many to many binary associations between these two classes.

Relationship between Mechanic and Service:
Furthermore, one mechanic can provide multiple types of services, and many mechanics can provide the same type of service. Therefore, we can see a many to many binary associations between these two classes.

UML Class Diagrams:



Objects:

James: Customer
firstName = “James” lastName = “Jones” gender = Gender.Male dateOfBirth = [16/04/1992] email = “james.jones@gmail.com” cellPhoneNumber = “816-897-9862”

Hans: Mechanic
firstName = “Hans” lastName = “K” gender = Gender.Male dateOfBirth = [21/03/1989] email = “hans.k@gmail.com” mechanicID= “00012891456”

AD-89034: Car
vehicleID = “AD-89034” make = “Nissan” model = “Altima “ yearOfModel = [2014] color = “Silver” engineType = EngineType.Gasoline numberOfDoors = NumberOfDoors.FOUR

OilReplacement: Service
name = “Oil Replacement” description = “This is the process of removing old dirty oil from the vehicle and replacing it with clean oil” price = 120.00 approximateDuration = 0.75 numberOfMechanics = NumberOfMechanics.ONE

Diagnostics: Service
name = " Diagnostics" description = "This is the process of process of checking a vehicle's systems and components to help identify issues and rectify them." price = 15.0 approximateDuration = 1.00 numberOfMechanics = NumberOfMechanics.ONE

OilFilterParts: Service
name = "OilFilterParts" description = "This is the of buying the necessary parts for an oil filter change." price = 35.0 approximateDuration = 0.25 numberOfMechanics = NumberOfMechanics.ONE

TireReplacement: Service
name = "Tire Replacement" description = "This is the process of changing a faulty/damaged tire" price = 100.0 approximateDuration = 1.00 numberOfMechanics = NumberOfMechanics.TWO

Tire: Service
name = "Tire" description = "This is the process of buying a new tire" price = 160.0 approximateDuration = 0.50 numberOfMechanics = NumberOfMechanics.ONE

Code for creating classes and the objects:

```
from enum import Enum

# Define Gender as an Enum
class Gender(Enum):
    MALE = 'Male'
    FEMALE = 'Female'

class Person: #define the class (this will be a parent class)
    # initialize the class objects with the specified attributes
    def __init__(self, firstName, lastName, gender, dateOfBirth, email):
        self._firstName = firstName
        self._lastName = lastName
        self._gender = gender
        self._dateOfBirth = dateOfBirth
        self._email = email

    # create setters and getters for each of the attributes defined above
    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 setGender(self, gender):
        self._gender = gender

    def getGender(self):
        return self._gender

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

    def getDateOfBirth(self):
        return self._dateOfBirth

    def setEmail(self, email):
```

```
        self._email = email

    def getEmail(self):
        return self._email

    def __str__(self):
        return "First Name: " + self._firstName + ", Last Name: " + self._lastName + ", Gender: " +
str(self._gender) + ", Date of Birth: " + str(self._dateOfBirth) + ", Email: " + self._email

class Customer(Person): #define the child class
    # initialize the class objects with the specified attributes
    def __init__(self, firstName, lastName, gender, dateOfBirth, email, cellPhoneNumber):
        super().__init__(firstName, lastName, gender, dateOfBirth, email)
        self._cellPhoneNumber = cellPhoneNumber

    # create setters and getters for each of the attributes defined above
    def setCellPhoneNumber(self, cellPhoneNumber):
        self._cellPhoneNumber = cellPhoneNumber

    def getCellPhoneNumber(self):
        return self._cellPhoneNumber

    def __str__(self):
        return super().__str__() + ", Cell Phone Number: " + self._cellPhoneNumber


class Mechanic(Person): #define the child class class
    # initialize the class objects with the specified attributes
    def __init__(self, firstName, lastName, gender, dateOfBirth, email, mechanicID):
        super().__init__(firstName, lastName, gender, dateOfBirth, email)
        self._mechanicID = mechanicID

    # create setters and getters for each of the attributes defined above
    def setMechanicID(self, mechanicID):
        self._mechanicID = mechanicID

    def getMechanicID(self):
        return self._mechanicID

    def __str__(self):
        return super().__str__() + ", Mechanic ID: " + self._mechanicID

# Define EngineType as an Enum
class EngineType(Enum):
    Gasoline = 'Gasoline'
```

```
    Diesel = 'Diesel'
    Electric = 'Electric'
    Hybrid = 'Hybrid'

class Vehicle: #define the class vehicle (this will be a parent class)
    # initialize the class objects with the specified attributes
    def __init__(self, vehicleID, make, model, yearOfModel, color, engineType):
        self._vehicleID = vehicleID
        self._make = make
        self._model = model
        self._yearOfModel = yearOfModel
        self._color = color
        self._engineType = engineType

    # create setters and getters for each of the attributes defined above
    def setVehicleID(self, vehicleID):
        self._vehicleID = vehicleID

    def getVehicleID(self):
        return self._vehicleID

    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 setYearOfModel(self, yearOfModel):
        self._yearOfModel = yearOfModel

    def getYearOfModel(self):
        return self._yearOfModel

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

    def getColor(self):
        return self._color
```

```
    def setEngineType(self, engineType):
        self._engineType = engineType

    def getEngineType(self):
        return self._engineType

    def __str__(self):
        return "Vehicle ID: "+str(self._vehicleID)+", Make: "+self._make+", Model: "+self._model+", Year of
Model: "+str(self._yearOfModel)+", Color: "+self._color+", Engine Type: "+self._engineType.value

#Define NumberOfDoors as enum
class NumberOfDoors(Enum):
    TWO = '2 Doors'
    FOUR = '4 Doors'

class Car(Vehicle): #define the child class class
    # initialize the class objects with the specified attributes
    def __init__(self, vehicleID, make, model, yearOfModel, color, engineType, numberOfDoors):
        super().__init__(vehicleID, make, model, yearOfModel, color, engineType)
        self._numberOfDoors = numberOfDoors

    # create setters and getters for each of the attributes defined above
    def setNumberOfDoors(self, numberOfDoors):
        self._numberOfDoors = numberOfDoors

    def getNumberOfDoors(self):
        return self._numberOfDoors

    def __str__(self):
        return "Vehicle ID: "+str(self._vehicleID)+", Make: "+self._make+", Model: "+self._model+", Year of
Model: "+str(self._yearOfModel)+", Color: "+self._color+", Engine Type: "+self._engineType.value+", Number
of Doors: "+self._numberOfDoors.value

#Define NumberOfMechanics as enum
class NumberOfMechanics(Enum):
    ONE = 1
    TWO = 2
    THREE = 3
    FOUR = 4
    FIVE_OR_MORE = 5

class Service: #define the class
    # initialize the class objects with the specified attributes
    def __init__(self, name, description, price, approximateDuration, numberOfMechanics):
        self._name = name
```

```
        self._description = description
        self._price = price
        self._approximateDuration = approximateDuration
        self._numberOfMechanics = numberOfMechanics

    # create setters and getters for each of the attributes defined above
    def setName(self, name):
        self._name = name

    def getName(self):
        return self._name

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

    def getDescription(self):
        return self._description

    def setPrice(self, price):
        self._price = price

    def getPrice(self):
        return self._price

    def setApproximateDuration(self, approximateDuration):
        self._approximateDuration = approximateDuration

    def getApproximateDuration(self):
        return self._approximateDuration

    def setNumberOfMechanics(self, numberOfMechanics):
        self._numberOfMechanics = numberOfMechanics

    def getNumberOfMechanics(self):
        return self._numberOfMechanics

    def __str__(self):
        return "Name: "+self._name+", Description: "+self._description+", Price: "+str(self._price)+",
Approximate Duration: "+str(self._approximateDuration)+" hours, Number of Mechanics Required:
"+str(self._numberOfMechanics.value)

#Customer object with the given attributes
customer = Customer(firstName="James", lastName="Jones", gender=Gender.MALE.name,
dateOfBirth="16/04/1992", email="james.jones@gmail.com", cellPhoneNumber="816-897-9862")
print(customer)
```

```
#Mechanic object with the given attributes
mechanic = Mechanic(firstName="Hans", lastName="K", gender=Gender.MALE.name, dateOfBirth=[21, 3, 1989], email="hans.k@gmail.com", mechanicID="00012891456")
print(mechanic)
#Car object with the given attributes
car = Car(vehicleID="AD-89034", make="Nissan", model="Altima", yearOfModel=[2014], color="Silver", engineType=EngineType.Gasoline, numberOfDoors=NumberOfDoors.FOUR)
print(car)
#Service object with the given attributes
service = Service(name="Oil Replacement", description="This is the process of removing old dirty oil from the vehicle and replacing it with clean oil", price=120.00, approximateDuration=0.75, numberOfMechanics=NumberOfMechanics.ONE)
print(service)
diagnostics = Service("Diagnostics", "This is the process of checking a vehicle's systems and components to help identify issues and rectify them.", 15.0, 1.00, NumberOfMechanics.ONE)
print(diagnostics)
oil_filter_parts = Service("OilFilterParts", "This is the of buying the necessary parts for an oil filter change.", 35.0, 0.25, NumberOfMechanics.ONE)
print(oil_filter_parts)
tire_replacement = Service("Tire Replacement", "This is the process of changing a faulty/damaged tire", 100.0, 1.00, NumberOfMechanics.TWO)
print(tire_replacement)
tire = Service("Tire", "This is the process of buying a new tire", 160.0, 0.50, NumberOfMechanics.ONE)
print(tire)
```

The output which is printing the objects created:

First Name: James, Last Name: Jones, Gender: MALE, Date of Birth: 16/04/1992, Email: james.jones@gmail.com, Cell Phone Number: 816-897-9862
First Name: Hans, Last Name: K, Gender: MALE, Date of Birth: [21, 3, 1989], Email: hans.k@gmail.com, Mechanic ID: 00012891456
Vehicle ID: AD-89034, Make: Nissan, Model: Altima, Year of Model: [2014], Color: Silver, Engine Type: Gasoline, Number of Doors: 4 Doors
Name: Oil Replacement, Description: This is the process of removing old dirty oil from the vehicle and replacing it with clean oil, Price: 120.0, Approximate Duration: 0.75 hours, Number of Mechanics Required: 1
Name: Diagnostics, Description: This is the process of checking a vehicle's systems and components to help identify issues and rectify them., Price: 15.0, Approximate Duration: 1.0 hours, Number of Mechanics Required: 1
Name: OilFilterParts, Description: This is the of buying the necessary parts for an oil filter change., Price: 35.0, Approximate Duration: 0.25 hours, Number of Mechanics Required: 1
Name: Tire Replacement, Description: This is the process of changing a faulty/damaged tire, Price: 100.0, Approximate Duration: 1.0 hours, Number of Mechanics Required: 2
Name: Tire, Description: This is the process of buying a new tire, Price: 160.0, Approximate Duration: 0.5 hours, Number of Mechanics Required: 1

Summary of learning:

I believe that this assignment helped me put my knowledge to test. I was able to reflect on the different topics we have learned and apply them to a specific real-world scenario. This was an assignment that I was extremely invested and excited about because it maps out the beginning of my journey in creating object-oriented programs. Upon learning about how to create classes and instances of classes I was able to deduce the scenario in the assignment and think about it from a programmer's perspective. I had so many different ideas of different implementations of the billing system however at the end I was able to choose the ideas that felt the most systematic and efficient (in terms of use cases).

Additionally, the use cases I wrote were effective because I went through different drafts of the use cases, and I tried to envision my place in that situation where I have to take my car to the garage. By putting myself in that scenario making the use case became easier because I could clearly see the different steps that I must take (scenarios of each use case) and where things can potentially go wrong (exceptions for each use case).

In addition, I was unsure whether or not to add the "make payment" use case however at the end I felt like the system created would not be complete or did not make sense to end without asking the customer to pay for their services.

Furthermore, in terms of relationships I was hesitant at first to identify the different relationships apart from inheritance because they were newly introduced. However, by examining the classes, objects, and the context it became easier to identify associations between several classes.

This assignment consisted of several drafts which is shown in the GitHub repository link. This represents my gradually progression on this assignment. I made sure that my work was precise and was constantly making changes to the syntax, format and overall structure of my assignment and the work I was producing.

I learned how turn my ideas of the billing system into a well-documented and running python code. I was able to create the different classes with their attributes and create instances of these the classes. Additionally, I learned that by having child classes inherit from parent classes it made writing the code easier, and more manageable. This is because I did not have to repeat huge chunks of code instead, I could use the super().

Github repository:

Link: https://github.com/aishalihwaidi/Assignment-1-ICS-220