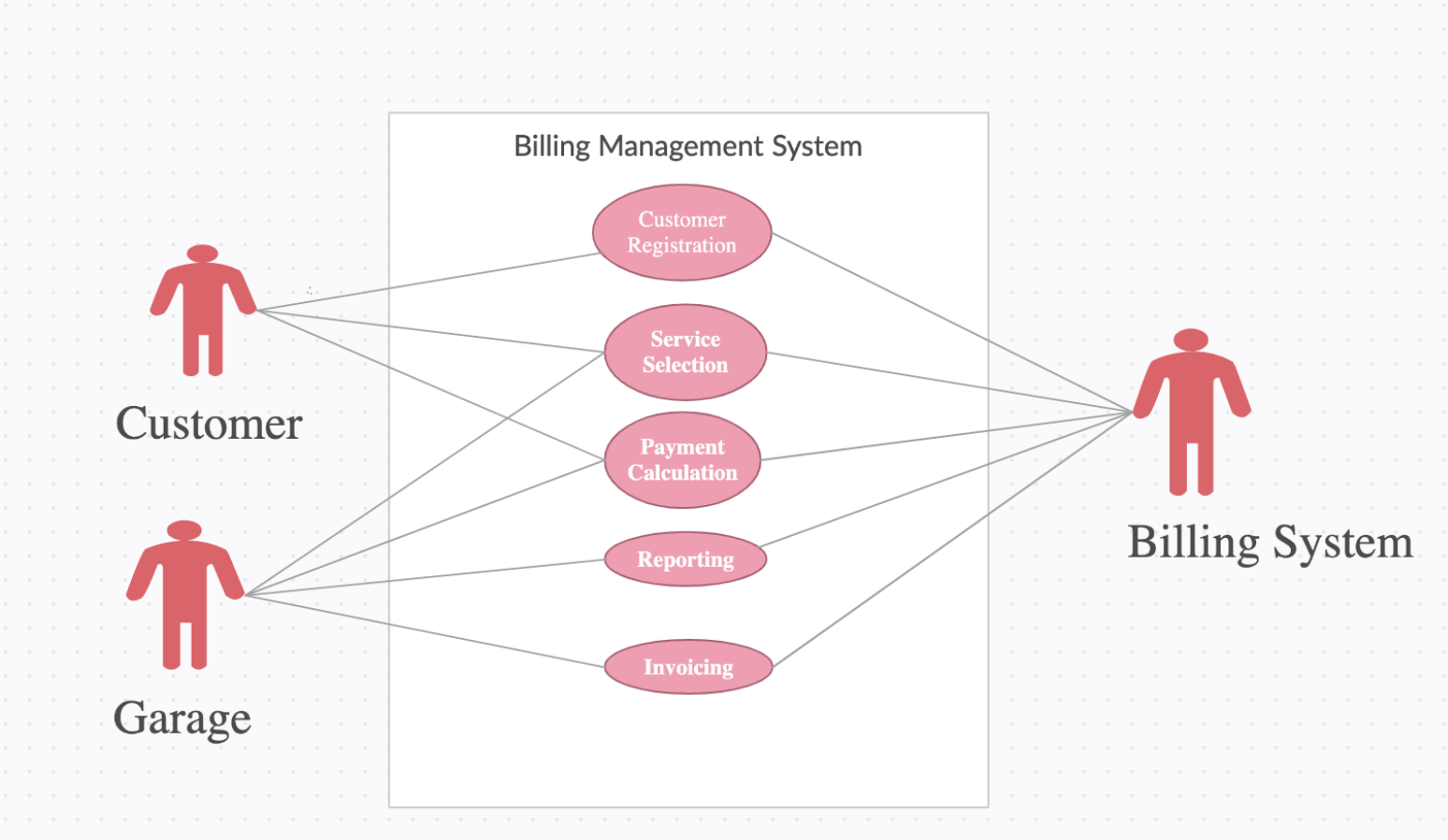
1.

1. Customer Registration: This use case involves allowing the garage to register new customers by collecting their personal information such as name, phone number, and vehicle details. The customer information can be stored in a database and used for future transactions and service history.
2. Service Selection: This use case involves allowing the customer to select the services they require, such as diagnostics, oil replacement, tire replacement, etc. The software must also provide the prices for each service and allow the customer to select multiple services if needed.
3. Payment Calculation: This use case involves calculating the total cost of the selected services, including the cost of parts and labor, taxes, and any discounts that apply. The software must be able to calculate the total amount due based on the customer's selection and provide an itemized bill.
4. Invoicing: This use case involves generating an invoice for the services provided, including a detailed breakdown of the services, taxes, discounts, and total amount. The invoice can be printed or sent electronically to the customer.
5. Reporting: This use case involves generating various reports, such as daily sales, customer information, service history, etc., for the garage management to make informed decisions. The software can provide insights into customer preferences, popular services, staff performance, and revenue trends. The reports can also help the garage make data-driven decisions on marketing, inventory, and staff management.

1. The "Customer" actor interacts with the "Billing System" through the following use cases: Customer Registration, Service Selection, Payment Calculation, Invoicing, and Reporting.
2. The "Garage" actor interacts with the "Billing System" through the following use cases: Service Selection, Payment Calculation, Invoicing, and Reporting.

3. The "Billing System" interacts with the actors and facilitates the following use cases: Customer Registration, Service Selection, Payment Calculation, Invoicing, and Reporting.

UML Use-case Diagram:



Use-case Descriptions:

Use Case:	Customer Registration

Trigger:	The user wants to register as a new customer with the garage
Preconditions:	The user has never registered with the garage before.
Main scenarios:	
1.	The user navigates to the garage's registration page.
2.	The user enters their personal information, including their full name, phone number, and email address, as well as their vehicle information, including the make, model, year, color, and vehicle identification number (VIN).
3.	The user submits the registration form.
4.	The system verifies the user's information and generates a unique customer ID.
5.	The system sends a confirmation message to the user with their customer ID.
Exceptions:	
4a.	1. The user's information is incomplete or invalid. 2. Cannot proceed, the system prompts the user to correct their information.
4b.	1. The user already has an account with the garage.

	2. The system prompts the user to log in instead.

Use Case:	Payment Calculation
Trigger:	The customer or garage wants to calculate the total cost of the selected services.
Preconditions:	The selected services are added to the customer or garage's invoice.
Main scenarios:	
1.	The customer or garage selects the "Calculate Payment" option from the billing system's menu
2.	The system applies the appropriate tax rate and any applicable discounts to the selected services
3.	The system calculates the final amount due, taking into account the cost of the services, taxes, and discounts.
4.	The system displays the final amount due to the customer or garage.
Exceptions:	
1a.	1. The selected services have not been added to the customer or garage's invoice. 2. The system prompts the user to add services to the invoice before calculating the payment.

2a.	1. The system encounters an error while applying tax or discounts. 2. The system prompts the user to contact customer support for assistance.
3a.	1. The system encounters an error while calculating the final amount due. 2. The system prompts the user to contact customer support for assistance.

Use Case:	Service Selection
Trigger:	The customer or garage wants to select the services they require.
Preconditions:	The customer or garage is registered with the billing system.
Main scenarios:	
1.	The customer or garage selects the "Select Services" option from the billing system's menu.
2.	The system displays a list of available services and their prices.
3.	The customer or garage selects the services they require by checking the corresponding checkboxes or entering the quantity for each service.

4.	The system validates the selected services and updates the invoice with the corresponding prices.
5.	The customer or garage reviews the updated invoice and confirms the selection of services.
6.	The system updates the service records and notifies the garage of the selected services.
Exceptions:	
3a.	1. The customer or garage selects an unavailable service. 2. Error, the system prompts the customer or garage to select available service.

Use Case:	Invoicing
Trigger:	The final amount due is calculated and displayed to the customer or garage.
Preconditions:	The final amount due is calculated and displayed to the customer or garage.
Main scenarios:	
1.	The customer or garage selects the "Generate Invoice" option from the billing system's menu.

2.	The system generates an invoice for the customer or garage based on the selected services and the final amount due.
3.	The system includes the customer or garage's details, such as name, address, and contact information, as well as the details of the services provided, the cost of each service, the total cost, any applicable taxes, and any discounts applied.
4.	The system sends the invoice to the customer or garage for payment.
Exceptions:	
3a.	1. If the system encounters an error generating the invoice. 2. It notifies the customer or garage and provides instructions for resolving the issue.
4a.	1. The invoice is not delivered to the customer or garage due to technical or other issues. 2. The system logs the issue and initiates a process to re-send or re-generate the invoice.

Use Case:	Reporting
Trigger:	The garage wants to generate a report on the services provided to customers and garages.
Preconditions:	The garage has provided services to at least one customer or garage.

Main scenarios:	
1.	The garage selects the "Generate Report" option from the billing system's menu.
2.	The system presents a range of report types to choose from, such as "Sales Report", "Service Report", "Customer Report", and "Vehicle Report".
3.	The garage selects the report type they wish to generate.
4.	The system prompts the garage to specify the parameters of the report, such as the time frame, the service category, or the customer/garage name.
5.	The garage enters the required parameters.
6.	The system generates the report based on the specified parameters.
7.	The system displays the generated report to the garage.
8.	The garage can choose to export the report in various formats, such as PDF, CSV, or Excel.
9.	The system exports the report in the selected format and saves it to the garage's device or sends it via email if requested.
Exceptions:	
1a.	1. There are no services provided to customers and garages. 2. The system displays an error message indicating that there is no data to generate a report.

2a.	1. The system encounters an error while generating the report. 2. The system prompts the user to contact customer support for assistance.

2.

Class and its attributes:

Person

- firstName (string)
- middleInitial (string)
- lastName (string)
- phoneNumber (string)
- gender (ENUM)

Customer

- firstName (string)
- middleInitial (string)
- lastName (string)
- phoneNumber (string)
- gender (ENUM)

Mechanic

- firstName (string)
- middleInitial (string)
- gender (ENUM)

- serviceAmount (int)
- serviceName (ENUM)

Vehicle

- make (ENUM)
- model (ENUM)
- color (string)
- year (int)
- vehicleID (string)

Service

- number (int)
- serviceName (ENUM)
- itemAmount(int)
- serviceAmount (int)
- currency (ENUM)
- price (float)

Payment

- total (float)
- taxes (float)
- discount (float)
- finalAmount (float)
- paymentDate (date)

Object:

Person:

object1:

- firstName: "James"
- lastName: "Jones"
- middleInitial: "W."
- phoneNumber: "816-897-9862"
- gender: Gender.Male

Customer:

object1:

- firstName: "James"
- lastName: "Jones"
- middleInitial: "W."
- phoneNumber: "816-897-9862"
- gender: Gender.Male

Mechanic:

object1:

- firstName: “Hans”
- middleInitial: “K”
- gender: Gender.Male
- serviceAmount: “5”
- serviceName: ServiceName.DIAGNOSTICS

object2:

- firstName: “Hans”
- middleInitial: “K”
- gender: Gender.Male
- serviceAmount: “5”
- serviceName: ServiceName.OIL_REPLACEMENT

object3:

- firstName: “Hans”
- middleInitial: “K”

- gender: Gender.Male
- serviceAmount: “5”
- serviceName: ServiceName.OIL_FILTER_PARTS

object4:

- firstName: “Hans”
- middleInitial: “K”
- gender: Gender.Male
- serviceAmount: “5”
- serviceName: ServiceName.TIRE_REPLACEMENT

object5:

- firstName: “Hans”
- middleInitial: “K”
- gender: Gender.Male
- serviceAmount: “5”
- serviceName: ServiceName.TIRE

Vehicle:

object1:

- make: Make.NISSAN
- model: Model.ALTIMA
- color: "Silver"
- year: “2014”
- vehicleID: "AD-89034"

Service:

object1:

- number: “1”
- serviceName: ServiceName.DIAGNOSTICS
- itemAmount: “1”
- currency: Currency.AED

- price: “15.0”

object2:

- number: “2”
- serviceName: ServiceName.OIL_REPLACEMENT
- itemAmount: “1”
- currency: Currency.AED
- price: “120.0”

object3:

- number: “3”
- serviceName: ServiceName.OIL_FILTER_PARTS
- itemAmount: “1”
- currency: Currency.AED
- price: “35.0”

object4:

- number: “4”
- serviceName: ServiceName.TIRE_REPLACEMENT
- itemAmount: “2”
- currency: Currency.AED
- price: “100.0”

object5:

- number: “5”
- serviceName: ServiceName.TIRE
- itemAmount: “2”
- currency: Currency.AED
- price: “160.0”

Payment:

object1:

- total: “451.5”
- taxes: “21.5”

- discount: “11.5”
- finalAmount: “440.0”
- paymentDate: “March 13,2022”

Behaviors of class:

Person

+getFirstName():String

+setFirstName(firstName:String)

+getMiddleInitial():String

+setMiddleInitial(middleInitial:String)

+getLastName():String

+setLastName(lastName:String)

+getPhoneNumber():String

+setPhoneNumber(phoneNumber:String)

+getGender():Gender

+setGender(gender:Gender)

Customer

+getFirstName():String

+setFirstName(firstName:String)

+getMiddleInitial():String

+setMiddleInitial(middleInitial:String)

+getLastName():String

+setLastName(lastName:String)

+getCellPhoneNumber():String

+setCellPhoneNumber(phoneNumber:String)

+getGender():Gender

+setGender(gender:Gender)

Mechanic

+getFirstName():String

+setFirstName(firstName:String)

+getMiddleInitial():String

+setMiddleInitial(middleInitial:String)

+getGender():Gender +setGender(gender:Gender)

+getServiceAmount():int

+setServiceAmount(serviceAmount:int)

+getServiceName():ServiceName

+setServiceName(serviceName:ServiceName)

Vehicle

+getMake():ENUM

+setMake(make:ENUM)

+getModel():ENUM

+setModel(model:ENUM)

+getColor():String

+setColor(color:String)

+getYear():int

+setYear(year:int)

+getVehicleID():String

+setVehicleID(vehicleID:String)

Service

+getNumber():int

+setNumber(number:int)

+getServiceName():ENUM

+setServiceName(serviceName:ENUM)

+getPrice():float

+setPrice(price:float)

Payment

+getTotal():float

+setTotal(total:float)

+getTaxes():float

+setTaxes(taxes:float)

+getDiscount():float

+setDiscount(discount:float)

+getFinalAmount():float

+setFinalAmount(finalAmount:float)

UML Class Diagram:

Class

Person
firstName (string) middleInitial (string) lastName (string) phoneNumber (string) gender (ENUM)
+getFirstName():String +setFirstName(firstName:String) +getMiddleInitial():String +setMiddleInitial(middleInitial:String) +getLastName():String +setLastName(lastName:String)

+getPhoneNumber():String
+setPhoneNumber(phoneNumber:String)
+getGender():Gender
+setGender(gender:Gender)

Object 1

James: Person
firstName: "James"
lastName: "Jones"
middleInitial: "W."
phoneNumber: "816-897-9862"
gender: Gender.Male

Object 2

Hans: Person
firstName: “Hans”
middleInitial: “K”
gender: Gender.Male

Class

Customer
firstName (string)
middleInitial (string)
lastName (string)
phoneNumber (string)
gender (ENUM)
+getFirstName():String
+setFirstName(firstName:String)
+getMiddleInitial():String

+setMiddleInitial(middleInitial:String)
+getLastName():String
+setLastName(lastName:String)
+getPhoneNumber():String
+setPhoneNumber(phoneNumber:String)
+getGender():Gender
+setGender(gender:Gender)

Object 1

James: Customer
firstName: "James"
lastName: "Jones"
middleInitial: "W."
phoneNumber: "816-897-9862"
gender: Gender.Male

Class

Mechanic
firstName (string)
middleInitial (string)
gender (ENUM)
serviceAmount (int)
serviceName (ENUM)
+getFirstName():String
+setFirstName(firstName:String)
+getMiddleInitial():String
+setMiddleInitial(middleInitial:String)
+getGender():Gender
+setGender(gender:Gender)

+getServiceAmount():int
+setServiceAmount(serviceAmount:int)
+getServiceName():ServiceName
+setServiceName(serviceName:ServiceName)

Object 1

Hans: Mechanic
firstName: “Hans”
middleInitial: “K”
gender: Gender.Male
serviceAmount: “5”
serviceName: ServiceName.DIAGNOSTICS

Object 2

Hans: Mechanic
firstName: “Hans”
middleInitial: “K”
gender: Gender.Male
serviceAmount: “5”
ServiceName.OIL_REPLACEMENT

Object 3

Hans: Mechanic
firstName: “Hans”
middleInitial: “K”
gender: Gender.Male
serviceAmount: “5”
ServiceName.OIL_FILTER_PARTS

Object 4

Hans: Mechanic
firstName: “Hans”
middleInitial: “K”
gender: Gender.Male
serviceAmount: “5”
ServiceName.TIRE_REPLACEMENT

Object 5

Hans: Mechanic
firstName: “Hans”
middleInitial: “K”
gender: Gender.Male
serviceAmount: “5”
serviceName: ServiceName.TIRE

Class

Vehicle
make (ENUM)
model (ENUM)
color (string)
year (int)
vehicleID (string)
+getMake():ENUM
+setMake(make:ENUM)
+getModel():ENUM
+setModel(model:ENUM)
+getColor():String
+setColor(color:String)
+getYear():int
+setYear(year:int)

+getVehicleID():String +setVehicleID(vehicleID:String)

Object 1

NissanAltima: Vehicle
Make= Make.NISSAN Model= Model.ALTIMA Color= "Silver" Year= “2014” vehicleID= "AD-89034"

Class

Service
number (int) serviceName (ENUM) itemAmount(int) currency (ENUM) price (float)
+getNumber():int +setNumber(number:int) +getServiceName():ENUM +setServiceName(serviceName:ENUM) +getPrice():float +setPrice(price:float)

Object 1

service1: Service
number: “1”
serviceName: ServiceName.DIAGNOSTICS
itemAmount: “1”
currency: Currency.AED
price: “15.0”

Object 2

Service2: Service
number: “2”
serviceName: ServiceName.OIL_REPLACEMENT
itemAmount: “1”
currency: Currency.AED
price: “120.0”

Object 3

Service3: Service
number: “3”
serviceName: ServiceName.OIL_FILTER_PARTS
itemAmount: “1”
currency: Currency.AED
price: “35.0”

Object 4

Service4: Service
number: “4”

serviceName: ServiceName.TIRE_REPLACEMENT itemAmount: “2” currency: Currency.AED price: “100.0”

Object 5

Service5: Service
number: “5” serviceName: ServiceName.TIRE itemAmount: “2” currency: Currency.AED price: “160.0”

Class

Payment
total (float) taxes (float) discount (float) finalAmount (float) paymentDate (date)
+getTotal():float +setTotal(total:float) +getTaxes():float +setTaxes(taxes:float) +getDiscount():float +setDiscount(discount:float) +getFinalAmount():float +setFinalAmount(finalAmount:float)

Object 1

payment1: Payment
total: “451.5”
taxes: “21.5”
discount: “11.5”
finalAmount: “440.0”
paymentDate: “March 13,2022”

supporting descriptions to explain the relationships :

Person and Customer:

- Customer class inherits from the Person class, as both classes have common attributes such as firstName, middleInitial, lastName, phoneNumber, and gender.

Mechanic and Person:

- Mechanic class inherits from the Person class, as both classes have common attributes such as firstName, middleInitial, and gender.

Service and Mechanic:

- Mechanic class has a behavior of providing services, so there is a composition relationship between Mechanic and Service classes, where a Mechanic can provide multiple Services to a Vehicle.

Vehicle:

- Vehicle class has a simple composition relationship with the Service class, as each Vehicle can have multiple Services performed on it.

Payment:

- Payment class has a simple association relationship with the Service class, as each Payment is associated with one or more Services performed on the Vehicle.

- there are no relationships between the Customer and Vehicle classes, as they represent different aspects of the Garage Services system.

These classes are related in the following ways:

- The Vehicle class has a composition relationship with the Mechanic class. This means that a Mechanic object is composed of a Vehicle object, because a Mechanic works on a specific Vehicle. The Mechanic class has a vehicle attribute that holds a reference to the Vehicle object.

- The Service class has a composition relationship with the Mechanic class. This means that a Mechanic object is composed of a list of Service objects, because a Mechanic performs various services on a Vehicle. The Mechanic class has a services attribute that holds a list of Service objects.

- The Payment class has a composition relationship with the Service class. This means that a Service object is composed of a Payment object, because a Service has associated payment information. The Service class has a payment attribute that holds a reference to the Payment object.

- The Customer and Mechanic classes have a inheritance relationship with the Person class. This means that both Customer and Mechanic classes inherit the attributes and behavior of the Person class, which represents a generalization of a person. The firstName, middleInitial, lastName, phoneNumber, and gender attributes are common to all three classes.

Respective relationships:

Person and Customer:

- Inheritance relationship: Customer inherits from Person, meaning that all the attributes and behaviors of Person are inherited by Customer.

Person and Mechanic:

- Inheritance relationship: Mechanic inherits from Person, meaning that all the attributes and behaviors of Person are inherited by Mechanic.

Vehicle and Mechanic:

- Association relationship: Mechanic uses Vehicle, meaning that Mechanic performs services on Vehicle.

Service and Mechanic:

- Association relationship: Mechanic uses Service, meaning that Mechanic performs services as defined in Service.

Payment and Service:

- Association relationship: Service uses Payment, meaning that Payment is used to make payment for the services provided by Service.

Explaining further:

1. Customer and Person: Customer class inherits from Person class. This means that the Customer class is a more specific version of the Person class, and it can access all the properties and methods of the Person class in addition to its own.

2. Mechanic and Person: Mechanic class also inherits from Person class, just like the Customer class. This means that the Mechanic class is a more specific version of the Person class, and it can access all the properties and methods of the Person class in addition to its own.

3. Service and Mechanic: There is a composition relationship between the Service and Mechanic classes. This means that the Mechanic class has a "has-a" relationship with the Service class, and it can use the properties and methods of the Service class to provide services to the customers.

4. Vehicle and Customer: There is an aggregation relationship between the Vehicle and Customer classes. This means that the Customer class has a "has-a" relationship with the Vehicle class, and it can own one or more vehicles. The Vehicle class can also exist independently of the Customer class.

5. Payment and Service: There is an association relationship between the Payment and Service classes. This means that the Payment class is associated with the Service class, and it can access the properties and methods of the Service class to calculate the payment amount. However, the Payment class does not inherit from the Service class, and it can exist independently of it.

3.

Python Classes Code:

```
from enum import Enum

class Gender(Enum):
    Male = "Male"
```

```
    Female = "Female"
    Other = "Other"

class Make(Enum):
    NISSAN = "Nissan"
    TOYOTA = "Toyota"
    HONDA = "Honda"
    FORD = "Ford"
    BMW = "BMW"
    MERCEDES = "Mercedes"

class Model(Enum):
    ALTIMA = "Altima"
    FOCUS = "Focus"
    GTR34 = "GTR34"
    CIVIC = "Civic"
    CAMRY = "Camry"
    MUSTANG = "Mustang"
    X6 = "X6"
    X5 = "X5"


class ServiceName(Enum):
    DIAGNOSTICS = "Diagnostics"
    OIL_REPLACEMENT = "Oil Replacement"
    OIL_FILTER_PARTS = "Oil Filter Parts"
    TIRE_REPLACEMENT = "Tire Replacement"
    TIRE = "Tire"

class Currency(Enum):
    AED = "AED"
```

```
    USD = "USD"
    EUR = "EUR"

class Person:
    def __init__(self, firstName, middleInitial, lastName, phoneNumber, gender):
        self.firstName = firstName
        self.middleInitial = middleInitial
        self.lastName = lastName
        self.phoneNumber = phoneNumber
        self.gender = gender

    def getFirstName(self):
        return self.firstName

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

    def getMiddleInitial(self):
        return self.middleInitial

    def setMiddleInitial(self, middleInitial):
        self.middleInitial = middleInitial

    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 getGender(self):
        return self.gender

    def setGender(self, gender):
        self.gender = gender

    def displayInfo(self):
        print("First Name: ", self.firstName, "Middle Initial: ", self.middleInitial, " Last Name: ",
self.lastName, " Phone Number: ", self.phoneNumber, " Gender: ", self.gender)



class Customer(Person):
    def __init__(self, firstName, middleInitial, lastName, phoneNumber, gender):
        super().__init__(firstName, middleInitial, lastName, phoneNumber, gender)
        self.firstName = firstName
        self.middleInitial = middleInitial
        self.lastName = lastName
        self.phoneNumber = phoneNumber
        self.gender = gender

    def getFirstName(self):
        return self.firstName

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

```
    def getMiddleInitial(self):
        return self.middleInitial

    def setMiddleInitial(self, middleInitial):
        self.middleInitial = middleInitial

    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 getGender(self):
        return self.gender

    def setGender(self, gender):
        self.gender = gender

    def displayInfo(self):
        super().displayInfo()
        print("First Name: ", self.firstName, "Middle Initial: ", self.middleInitial, " Last Name: ",
self.lastName, " Phone Number: ", self.phoneNumber, " Gender: ", self.gender)


class Mechanic(Person):
```

```
def __init__(self, firstName, middleInitial, gender, serviceAmount, serviceName):
    super().__init__(firstName, middleInitial, None, None, gender)
    self.firstName = firstName
    self.middleInitial = middleInitial
    self.gender = gender
    self.serviceAmount = serviceAmount
    self.serviceName = serviceName

def getFirstName(self):
    return self.firstName

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

def getMiddleInitial(self):
    return self.middleInitial

def setMiddleInitial(self, middleInitial):
    self.middleInitial = middleInitial

def getGender(self):
    return self.gender

def setGender(self, gender):
    self.gender = gender

def getServiceAmount(self):
    return self.serviceAmount

def setServiceAmount(self, serviceAmount):
    self.serviceAmount = serviceAmount
```

```
    def getServiceName(self):
        return self.serviceName

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

    def displayInfo(self):
        super().displayInfo()
        print("First Name: ", self.firstName, "Middle Initial: ", self.middleInitial, " Gender: ",
self.gender, "Service Amount: ", self.serviceAmount, "Service Name: ", self.serviceName)



class Vehicle:
    def __init__(self, make, model, color, year, vehicleID):
        self.make = make
        self.model = model
        self.color = color
        self.year = year
        self.vehicleID = vehicleID

    def getMake(self):
        return self.make

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

    def getModel(self):
        return self.model
```

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

    def getColor(self):
        return self.color

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

    def getYear(self):
        return self.year

    def setYear(self, year):
        self.year = year

    def getVehicleID(self):
        return self.vehicleID

    def setVehicleID(self, vehicleID):
        self.vehicleID = vehicleID

    def displayInfo(self):
        print("Make: ", self.make, " Model: ", self.model, " Color: ", self.color, " Year: ", self.year,
" Vehicle ID: ", self.vehicleID)



class Service:
    def __init__(self, number, serviceName, itemAmount, currency, price):
        self.__number = number
        self.__serviceName = serviceName
        self.__itemAmount = itemAmount
```

```
        self.__currency = currency
        self.__price = price

    def getNumber(self):
        return self.__number

    def setNumber(self, number):
        self.__number = number

    def getServiceName(self):
        return self.__serviceName

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

    def getItemAmount(self):
        return self.__itemAmount

    def setItemAmount(self, itemAmount):
        self.__itemAmount = itemAmount

    def getCurrency(self):
        return self.__currency

    def setCurrency(self, currency):
        self.__currency = currency

    def getPrice(self):
        return self.__price

    def setPrice(self, price):
```

```
        self.__price = price

    def displayInfo(self):
        print("Number: ", self.__number, " Service Name ", self.__serviceName, "Item Amount: ",
self.__itemAmount, "Currency: ", self.__currency, "Price: ", self.__price)



class Payment:
    def __init__(self, total, taxes, discount, finalAmount, paymentDate):
        self.__total = total
        self.__taxes = taxes
        self.__discount = discount
        self.__finalAmount = finalAmount
        self.__paymentDate = paymentDate

    def getTotal(self):
        return self.__total

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

    def getTaxes(self):
        return self.__taxes

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

    def getDiscount(self):
        return self.__discount

    def setDiscount(self, discount):
```

```
        self.__discount = discount

    def getFinalAmount(self):
        return self.__finalAmount

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

    def getPaymentDate(self):
        return self.__paymentDate

    def setPaymentDate(self, paymentDate):
        self.__paymentDate = paymentDate

    def displayInfo(self):
        print("Total: ", self.__total , " Taxes: ", self.__taxes , " Discount: ", self.__discount , " Final
Amount: ", self.__finalAmount , " Payment Date: ", self.__paymentDate )



person1 = Person(firstName="James", lastName="Jones", middleInitial="W",
phoneNumber="123-456-7890", gender=Gender.Male)



customer1 = Customer(firstName="James", lastName="Jones", middleInitial="W.",
phoneNumber="816-897-9862", gender=Gender.Male)

mechanic1 = Mechanic(firstName="Hans", middleInitial="K", gender= Gender.Male,
serviceAmount=1, serviceName=ServiceName.DIAGNOSTICS)
mechanic2 = Mechanic(firstName="Hans", middleInitial="K", gender=Gender.Male,
serviceAmount=1, serviceName=ServiceName.OIL_REPLACEMENT)
```

```
mechanic3 = Mechanic(firstName="Hans", middleInitial="K", gender=Gender.Male,
serviceAmount=1, serviceName=ServiceName.OIL_FILTER_PARTS)
mechanic4 = Mechanic(firstName="Hans", middleInitial="K", gender=Gender.Male,
serviceAmount=2, serviceName=ServiceName.TIRE_REPLACEMENT)
mechanic5 = Mechanic(firstName="Hans", middleInitial="K", gender=Gender.Male,
serviceAmount=2, serviceName=ServiceName.TIRE)


vehicle1 = Vehicle(make=Make.NISSAN, model=Model.ALTIMA, color="Silver",
year="2014", vehicleID="AD-89034")


service1 = Service(number="1", serviceName=ServiceName.DIAGNOSTICS, itemAmount="1",
currency=Currency.AED, price="15.0")
service2 = Service(number="2", serviceName=ServiceName.OIL_REPLACEMENT,
itemAmount="1", currency=Currency.AED, price="120.0")
service3 = Service(number="3", serviceName=ServiceName.OIL_FILTER_PARTS,
itemAmount="1", currency=Currency.AED, price="35.0")
service4 = Service(number="4", serviceName=ServiceName.TIRE_REPLACEMENT,
itemAmount="2", currency=Currency.AED, price="100.0")
service5 = Service(number="5", serviceName=ServiceName.TIRE, itemAmount="2",
currency=Currency.AED, price="160.0")


payment1 = Payment(total="451.5", taxes="21.5", discount="11.5", finalAmount="440.0",
paymentDate="March 13, 2022")


person1.displayInfo()

customer1.displayInfo()
```

```
mechanic1.displayInfo()
mechanic2.displayInfo()
mechanic3.displayInfo()
mechanic4.displayInfo()
mechanic5.displayInfo()

vehicle1.displayInfo()

service1.displayInfo()
service2.displayInfo()
service3.displayInfo()
service4.displayInfo()
service5.displayInfo()

payment1.displayInfo()
```

Output:

```
First Name: James Middle Initial: W Last Name: Jones Phone Number: 123-456-7890 Gend
er: Gender.Male
First Name: James Middle Initial: W. Last Name: Jones Phone Number: 816-897-9862 Gend
er: Gender.Male
First Name: James Middle Initial: W. Last Name: Jones Phone Number: 816-897-9862 Gend
er: Gender.Male
First Name: Hans Middle Initial: K Last Name: None Phone Number: None Gender: Gende
r.Male
First Name: Hans Middle Initial: K Gender: Gender.Male Service Amount: 1 Service Name:
ServiceName.DIAGNOSTICS
```

First Name: Hans Middle Initial: K Last Name: None Phone Number: None Gender: Gender.Male

First Name: Hans Middle Initial: K Gender: Gender.Male Service Amount: 1 Service Name: ServiceName.OIL_REPLACEMENT

First Name: Hans Middle Initial: K Last Name: None Phone Number: None Gender: Gender.Male

First Name: Hans Middle Initial: K Gender: Gender.Male Service Amount: 1 Service Name: ServiceName.OIL_FILTER_PARTS

First Name: Hans Middle Initial: K Last Name: None Phone Number: None Gender: Gender.Male

First Name: Hans Middle Initial: K Gender: Gender.Male Service Amount: 2 Service Name: ServiceName.TIRE_REPLACEMENT

First Name: Hans Middle Initial: K Last Name: None Phone Number: None Gender: Gender.Male

First Name: Hans Middle Initial: K Gender: Gender.Male Service Amount: 2 Service Name: ServiceName.TIRE

Make: Make.NISSAN Model: Model.ALTIMA Color: Silver Year: 2014 Vehicle ID: AD-89034

Number: 1 Service Name ServiceName.DIAGNOSTICS Item Amount: 1 Currency: Currency.AED Price: 15.0

Number: 2 Service Name ServiceName.OIL_REPLACEMENT Item Amount: 1 Currency: Currency.AED Price: 120.0

Number: 3 Service Name ServiceName.OIL_FILTER_PARTS Item Amount: 1 Currency: Currency.AED Price: 35.0

Number: 4 Service Name ServiceName.TIRE_REPLACEMENT Item Amount: 2 Currency: Currency.AED Price: 100.0

Number: 5 Service Name ServiceName.TIRE Item Amount: 2 Currency: Currency.AED Price: 160.0

Total: 451.5 Taxes: 21.5 Discount: 11.5 Final Amount: 440.0 Payment Date: March 13, 2022

Github repository link, with access made public:

https://github.com/fatmanassiri/Assignment-1

Summary of Learnings:

I learned a lot during this assignment, and have done a lot of mistakes and errors, but these ended up being re-made into a better one. I learned so much about object oriented programming, which is the OOP, where is it is a programming paradigm that uses objects to represent and interact with data and functionality in a program. I have made Use Cases diagram and descriptions, and learned throughout it the actors, the use cases, and how to create a diagram, and based on the diagram I created a description, which then helped me with my classes and objects to identify easier.

classes are templates or blueprints for creating objects in OOP. They define the properties and methods that objects of that class will have. Which is why it's playing an important rule for me, to make my python code much more easier by looking at the diagram and descriptions. objects are instances of a class. They have their own unique data and can perform actions or behaviors defined by the class.

I have also learned a lot about inheritance, I was absent at the class, which made it very hard for me, but with the help of my lab class professor and my peers, I managed to create two more classes to add an inheritance. But it took some struggling to manage it perfectly.

Inheritance is a feature of OOP that allows a class to inherit properties and methods from another class. The class that inherits from another is called a subclass or derived class, while the class it inherits from is called a superclass. The super() function is used to call a method in a superclass from a subclass. I used super from what I have learned to create a parent and a child inheritance, which I somehow created a class to make it more sense.

I learned about encapsulation which is the practice of hiding the implementation details of a class and only exposing the necessary information. Where we use init method and if we use double underscores for private or we can make it public, whichever made sense. That is what abstraction is, its the practice of only exposing the relevant information and functionality of a class, while hiding the unnecessary details.

I have also learned getters and setters which are methods that are used to access or modify the values of an object's properties, respectively.

And then I have managed to create a perfect running code, by print a function info, which prints the whole class, rather than printing each attribute of an object.

Based on LOs:

LO1_OOAD: Analyze and design software that map real-world entities and relationships using Unified Modelling Language (UML) notations.

- I learned how to analyze real world entities and relationships and map them into python coding using UML notations.
- I also learned how to identify, describe, and model classes, attributes, operations, and associations in UML.
- And learned how to use UML diagrams such as class diagrams and use case diagrams to visualize and type in python coding.

LO2_OOProgramming: Create working object-oriented programs in a computer language that are well- structured, error free, and can solve computational problems.

- I learned how to design and implement object-oriented programs in python coding.

- I also learned how to use fundamental concepts of object-oriented programming such as classes, objects, and inheritance.
- I learned how to write well-structured, error-free, and efficient code that can solve all the errors, and asks what was needed from the assignment.