ICS220 22873 Program. Fund.

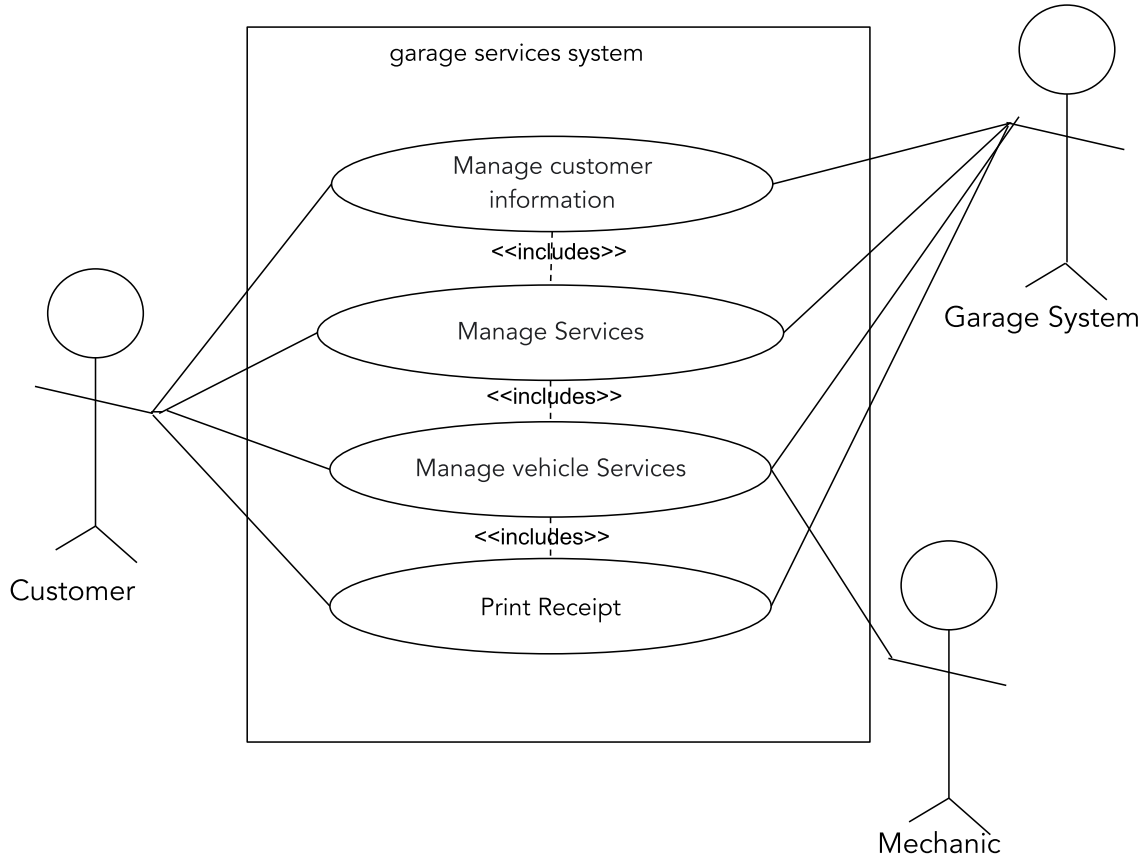
Assignment 1: Software Modelling UML Use Case and UML Class diagrams

Prof Kuhail

Wed 1 Mar 2023

Shamsa Rasehd

1. Identify the use cases for the software. Draw the **UML use-case diagram **and include supporting use-case descriptions. At-least 3 scenarios must be identified.



From the diagram, we can see three actors.

1) The customer is the vehicle owner who has brought their vehicle in for service. They use the garage's system to make appointments, bring and retrieve their vehicles, and pay for work.

2) The garage system is the group or system that runs the garage and does things like schedule appointments, assign mechanics, bill customers, and more.

3) The mechanic is the person who performs the actual repair or maintenance work on the customer's vehicle.

At the same time, we can see the whole system " the rectangle which is the garage services system " in the garage service system there are 4 ovals which represent the 4 use cases garage system used when a customer comes to the garage.

1) Manage customer information: In this scenario, the garage system communicates with the client to collect the data it needs to carry out the service. The consumer plays a supporting role in the garage system in this scenario. The garage

management system must gather information such as the customer's name, phone number, and vehicle specifics.

2) Manage services: The customer and the garage system are the main actors in this scenario. Clients make service requests, like oil changes and brake installations, through the garage's automated system. The garage system suggests a list of the service for the customer containing the price and the service name. The consumer can add or remove the services and it is up to them to decide whether or not to proceed with the service. after choosing the services the garage system verifies the date for the customer to bring the vehicle.

3) Manage vehicle service: In this scenario, the customer, the garage management system, and the mechanic all play roles in the management of vehicle servicing. When a customer brings their vehicle in for service, the garage management system determines who will work on the car and sends them to them. A mechanic's job is to check out a vehicle, figure out what's wrong with it, and fix or keep it in good shape. Once the service is finished, the garage's system will verify the service and then send the customer a notification and a comprehensive report.

4) Print Recpet: In this scenario, the customer and garage management system are the main actors, and the use case is to print a receipt. The consumer receives a receipt from the garage's system that itemizes the services rendered and the associated costs after the work is finished. The customer makes a payment for the work done, and the garage management system issues them a receipt for their records.

In summary, the relationships between the actors are as follows:

- In all scenarios, the garage system is the main player because it is in charge of the service and has to communicate with the other actors.
- In each scenario, the consumer must engage with the garage system in some way to either supply data, make a service request, or make a transactional payment.
- In the "Manage vehicle services" use case, the mechanic communicates with the garage system to do the necessary servicing and repairs for the customer's vehicle.

Use case 1: Manage customer information

Use Case:	Manage customer information
Trigger:	The system manages customer information and registers him/her in the system.
Main scenarios:	1. The customer specifies his/her personal information first name, last name, EID, email, and phone number.

	2. The garage system verifies the customer's information
	3. The garage system asks the customer to provide the vehicle information, make, year, model, type, color, and ID.
	4. The garage system verifies the vehicle information.
2a	- The customer information is incorrect. - Ex. (The customer ID is invalid, The customer number is invalid…) - The use case ends, and an error message is communicated.
3a	- The vehicle information is incorrect. - The use case ends, and an error message is communicated

Use Case:	Manage Services
Trigger:	The customer wants to add a garage service.
precondition	- The customer is registered in the system.
Main scenario:	1. Clients make service requests.
	2. The garage system displays a list of existing services.
	3. The customer selects the wanted services from the list.
	4. the customer can choose and remove unwanted services from the list.
	5. The garage system updates the list.
	6. The garage system verifies the list
	8. The garage system gives the customer date to bring the car.
	9. The garage system verifies the date.
6a	- The user wants to add one extra service. - The user wants to remove the service. - The use case ends, and an error message is communicated.

9a	- If the date is not suitable for the customer. - If the mechanic is not available on that date. - The use case ends, and an error message is communicated.

Use Case:	Manage vehicle Services
Trigger:	- A customer brings a vehicle for service and the garage system generates a list of the done services with mechanic info.
precondition	- A list of wanted services is generated and the customer book a day to bring the car.
Main scenario:	1. The garage system initiates a new service request for the customer's car.
	2. The garage system assigns the request to the vehicle ID.
	3. A mechanic is sent to the service request by the garage system .
	4. The mechanic checks the service request.
	5. The mechanic performs the requested service on the vehicle.
	6. The mechanic logs the accomplished service with his name, email, and phone number in the system.
	7. The mechanic confirms the service and tells the garage system that the service has been completed and writes a brief description of the service and how long it takes with the price.
	8. The garage system reviews the completed service request and add the mechanic's name to the list of the done services.
	9. The garage system tells the customer that the vehicle is ready for pickup.
8a	- if there is a service that has not been accomplished the use case ends, and

	an error message is communicated.

Use Case:	Print Receipt
Trigger:	- The customer wants to pay for the services that have been done.
precondition	- A list of the done services is generated with the mechanic's name.
Main scenario:	1. The system tells the user to enter any discount code that is available.
	2. User enters discount code, if available.
	3. If the code is valid, the system takes the discount off the total.
	4. System adds taxes based on the total cost.
	5. The system shows the user the total cost with taxes and any discounts.
	6. the customer chose the payment method
	7. the garage system verify the payment method.
	8. The system prints the receipt with the customer name, phone number, vehicle information, date, mechanic's info, list of the accomplished services with each service price, the taxes, the discount, And the total cost.
3a	If the code is invalid the use case ends, and an error message is communicated.
7a	If the payment method is invalid the use case ends, and an error message is communicated.

2. Identify the objects and their respective classes. Draw the **UML class diagrams **and include supporting descriptions to explain the relationships. At-least 4 classes and respective relationships must be identified.

1. Person:

   Attributes:

   - firstName: string
   - lastName: staring

- phoneNumber: string
- email: string

2. Customer:
   I consider the customer as a class because it represents a distinct entity with a specific set of attributes and behaviors. A customer class inherits the person class attributes, so it considers a child. and it contains one extra attribute which is the customer EID.
   Attributes:
   - firstName: string
   - lastName: staring
   - phoneNumber: string
   - customerEID: string
   - Email: string

3. Mechanic:
   A Mechanic class inherits the person class attributes, so it considers a child. and it contains TWO extra attributes which are employee ID and year of Experience.
   Attributes:
   - firstName: string
   - lastName: staring
   - phoneNumber: string
   - email: string
   - employeeID: string
   - experience: int

4. vehicle:
   I consider a Vehicle as a class because it is an entity that has certain attributes (such as make, model, year, color, etc.) that can be modeled in a program.
   Attributes:
   - make: string
   - model: string
   - Year: string
   - color: Enum
   - ID: string

There is a direct line of inheritance between the three linked classes, Human, Customer, and Mechanic.

Common attributes and methods shared by the Customer and Mechanic classes include firstName, lastName, phone number, and email, all of which are inherited from the parent or base class, the Person class. These shared characteristics and operations are inherited from the Person class and used by both the Customer and the Mechanic classes.

5. Service:
   I consider Service as a class because it stands for a concept or entity in the problem domain that has properties and actions. In the context of a repair shop, a Service class would hold information about the services offered by the mechanic in the garage, such as the name, description, price, and length of time. By making a Service class, I can put all of the relevant data and operations for a service into a single entity. This makes the code more modular, easier to maintain, and reusable.

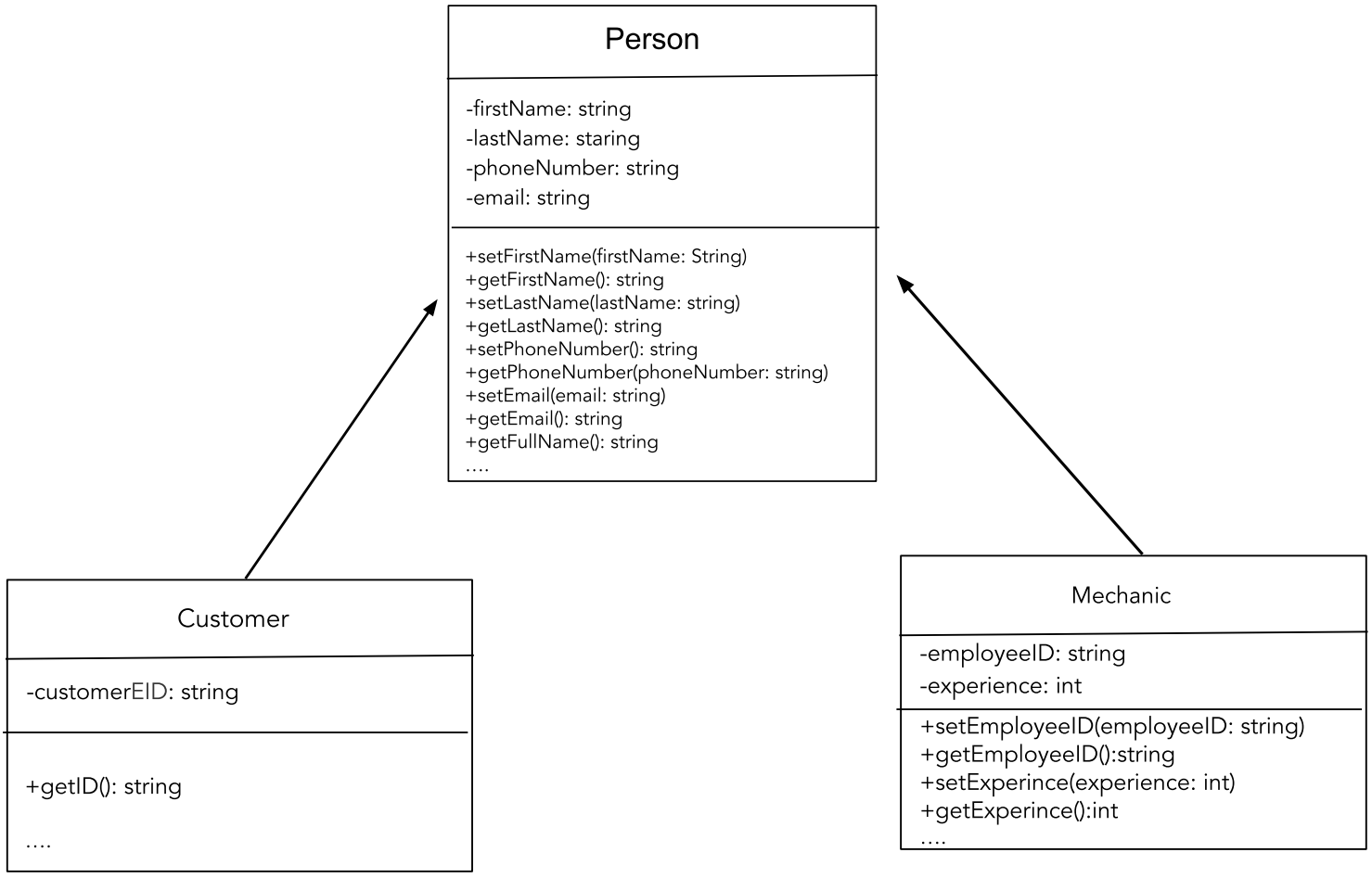
   Attributes:
   - name: string
   - description: string
   - duration: string
   - price: float

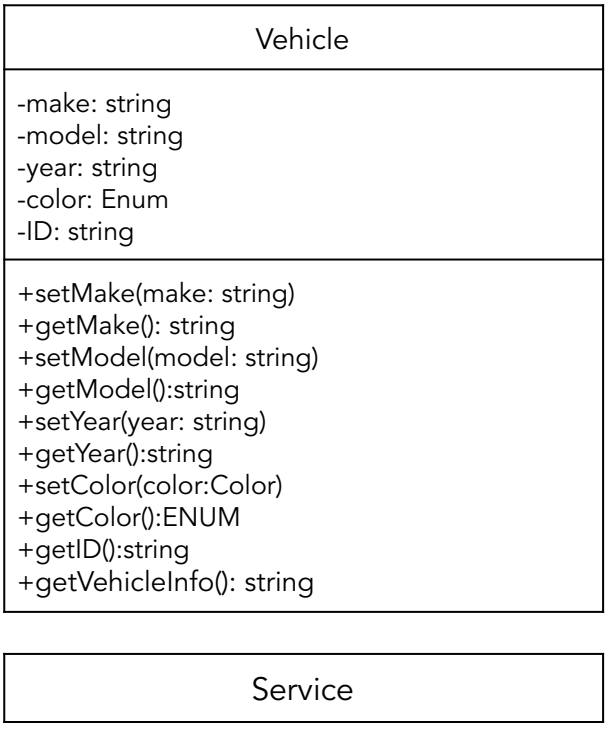
6. Receipt:
   I considered the Receipt as a class because it represents a collection of information and actions related to a customer's transaction. In other words, it is a blueprint that defines the properties and behavior of a receipt. A Receipt class can have attributes such as customer information, data, services provided, taxes, discounts, and total amount. It can also have methods such as calculating the total amount and printing the receipt. By making a Receipt class, I can encapsulate the data and behavior related to receipts. This makes it easier to manage the code and use it in other places. I can also make multiple copies of the Receipt class to handle different transactions.

   Attributes:
   - customer: Customer
   - mechanic: string
   - vehicle: Vehicle
   - services: List[Services]
   - date: date
   - taxes: float
   - discount: float

**UML Class **Diagram and Description:

There is a direct line of inheritance between the three linked classes, Human, Customer, and Mechanic.
Common attributes and methods shared by the Customer and Mechanic classes include firstName, lastName, phone number, and email, all of which are inherited from the parent or base class, the Person class. These shared characteristics and operations are inherited from the Person class and used by both the Customer and the Mechanic classes.


Person
-firstName: string
-lastName: staring
-phoneNumber: string
-email: string
+setFirstName(firstName: String)
+getFirstName(): string
+setLastName(lastName: string)
+getLastName(): string
+setPhoneNumber(): string
+getPhoneNumber(phoneNumber: string)
+setEmail(email: string)
+getEmail(): string
+getFullName(): string
....
Customer
-customerEID: string
+getID(): string
....
Mechanic
-employeeID: string
-experience: int
+setEmployeeID(employeeID: string)
+getEmployeeID():string
+setExperince(experience: int)
+getExperince():int
....


Vehicle
-make: string
-model: string
-year: string
-color: Enum
-ID: string
+setMake(make: string)
+getMake(): string
+setModel(model: string)
+getModel():string
+setYear(year: string)
+getYear():string
+setColor(color:Color)
+getColor():ENUM
+getID():string
+getVehicleInfo(): string

Service

-name:string -description:string -duration : string -price:float
+setName(serviceName:string) +getName():string +setDescription(description:string) +getDescription():string +setDuration(duration : string) +getDuration():string +setPrice(price:float) +getPrice():float

Receipt
-customer :Customer -cellPhoneNumber: string -mechanic: string -vechicle: Vehicle -services: List[Services] -date:string -taxes: float -discount: float
+getCustomer(): Customer +getPhoneNumber(): string +setMechanic(mechanic:string) +getMechanic():string +setDate(date:string) +getDate():date +setTaxes(taxes:float) +getTaxes():float +setDiscount(discount:float) +getDiscount(): float +Total(): float (sum(Service Price) +Toatl+=taxes +Total-=discount

Python code :

```
from enum import Enum

class Person:
```

```
    def __init__(self, firstName, lastName, phoneNumber, email):
        self.__firstName = firstName
        self.__lastName = lastName
        self.__phoneNumber = phoneNumber
        self.__email = email

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

    def getPhoneNumber(self):
        return self.__phoneNumber

    def setEmail(self, email):
        self.__email = email

    def getEmail(self):
        return self.__email

    def getFullName(self):
        return self.__firstName + " " + self.__lastName

    def __str__(self):
        return f"Name: {self.getFullName()}\n Cell Phone Number:
{self.__phoneNumber}\nEmail: {self.__email}"



class Customer(Person):
    def __init__(self, firstName, lastName, phoneNumber, customerEID,
email):
        super().__init__(firstName, lastName, phoneNumber, email)
```

```
        self.__customerEID = customerEID

    def setCustomerEID(self, customerEID):
        self.__customerEID = customerEID

    def getCustomerEID(self):
        return self.__customerEID

    def __str__(self):
        return f"{super().__str__()}\ncustomerEID: {self.__customerEID}"

class Mechanic(Person):
    def __init__(self, firstName, lastName, phoneNumber, email,
employeeID, experience):
        super().__init__(firstName, lastName, phoneNumber, email)
        self.__employeeID = employeeID
        self.__experience = experience

    def setEmployeeID(self,employeeID):
      self.__employeeID=employeeID
    def getEmployeeID(self):
      return self.__employeeID
    def setExperience(self,experience):
      self.__experience=experience
    def getExperience(self):
      return self.__experience

class Color(Enum):
    white = 1
    Black = 2
    Silver = 3
    Red = 4


class Vehicle:
    def __init__(self, make, model, year, color, ID):
        self.__make = make
        self.__model = model
        self.__year = year
        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 setYear(self, year):
        self.__year = year

    def getYear(self):
        return self.__year

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

    def getColor(self):
        return self.__color

    def getID(self):
        return self.__ID

    def getVehicleInfo(self):
        return self.__make + " " + self.__model + " " + str(self.__year)


class Service:
    def __init__(self, name, description, duration, price):
        self.__name = name
        self.__description = description
        self.__duration = duration
        self.__price = price

    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 setDuration(self, duration):
        self.__duration = duration

    def getDuration(self):
        return self.__duration

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

    def getPrice(self):
        return self.__price


class Receipt:
    def __init__(self,Customer,mechanic,vehicle, services, date, taxes,
discount):
        self.__customer=customer
        self.__mechainc=mechanic
        self.__vehicle=vehicle
        self.__services=services
        self.__date =date
        self.__taxes=taxes
        self.__discount=discount

    def setDate(self,date):
        self.__date = date
    def getDate(self):
        return self.__date
    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 total(self):
        total=sum(Service.getPrice() for Service in self.__services)
        total+=self.__taxes
        total-=self.__discount
        return total

    def PrintReceipt(self):
        print("customer info : ")
        print(customer)
        print("Date:",self.__date)
        print('                ') # to create space and organize the
Receipt

        print("Mechanic info : ")
        print(mechanic)
        print('                ') # to create space and organize the
Receipt

        print("Vehicle Type:",self.__vehicle.getVehicleInfo())
        print("Vehicle color: ",self.__vehicle.getColor())
        print("Vehicle ID:",self.__vehicle.getID())

        print('                ')
        print("Services:",)

        for service in self.__services:
            print(service.getName() , ".....................",
service.getPrice(),"AED",".","duration:", service.getDuration())
        print("Taxes:", self.__taxes)
        print("Discount:", self.__discount)
        print("Total:", self.total())


# Create a Customer object
customer = Customer("James", "Jones", "816-897-9862","1234567890",
"jamesjones@example.com")

# Create a mechanic object
mechanic = Mechanic("Hans", "K", "0987654321", "janesmith@email.com",
"M001", "5 years")
# Create a Vehicle object
vehicle = Vehicle("Nissan", "Altima", 2014, Color.Silver.name,
"AD-89034")
```

```
# Create a Service object
diagnostics = Service("1.Diagnostics","looking at the care and
dlajd","5 hours", 15.0)
oilReplacement = Service("2.Oil Replacement","changing the oil ","8 h",
120.49)
oilFilterParts = Service("3.Oil Filter Parts","changing oil Filter
Parts","8 h", 35)
tireReplacement =Service("4.Tire Replacement 2 ","replace the tire ","5
h", 100)
tire= Service("5.Tire Replacement (2) ","replace the tire ","5 h", 160)


# Create a Receipt object
receipt = Receipt(customer, mechanic,vehicle=vehicle,
services=[diagnostics,oilReplacement,oilFilterParts,tireReplacement,tir
e], date="March 13, 2022", taxes=51.5, discount=11.5)



# Print the receipt
receipt.PrintReceipt()
```

Output:

```
customer info :
Name: James Jones
 Cell Phone Number: 816-897-9862
Email: jamesjones@example.com
customerEID: 1234567890
Date: March 13, 2022

Mechanic info :
Name: Hans K
 Cell Phone Number: 0987654321
Email: janesmith@email.com

Vehicle Type: Nissan Altima 2014
Vehicle color:  Silver
Vehicle ID: AD-89034
```

```
Services:
1.Diagnostics ..................... 15.0 AED . duration: 5 hours
2.Oil Replacement ..................... 120.49 AED . duration: 8 h
3.Oil Filter Parts ..................... 35 AED . duration: 8 h
4.Tire Replacement 2  ..................... 100 AED . duration: 5 h
5.Tire Replacement (2)  ..................... 160 AED . duration: 5 h
Taxes: 51.5
Discount: 11.5
Total: 470.49
```

Github repository link:

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

References:

https://docs.python.org/3/tutorial/classes.html#instance-objects
https://pynative.com/python-encapsulation/
https://www.tutorialspoint.com/getter-and-setter-in-python
https://realpython.com/python3-object-oriented-programming/

When I was working on my assignment to print a receipt, I needed to find a way to print attributes from different classes to print the receipt and my attributes were privet. I did some research and found some great resources that explained how to use getter functions to get information from different classes.

The Python documentation on object-oriented programming was one of the things I found that could help me. I learned how to define classes with getter functions in the "9. Classes" section. In particular, the "9.3.3. Method Objects" section talked about how to use methods (including getters) to get information from instances of a class.

I also found the GeeksforGeeks tutorial on Python classes Section 3 ("Access Modifiers and Encapsulation in Python Classes"), and the PYnative Python Programming Encapsulation in Python very useful, I learned about encapsulation and how getter functions can be used to show information while hiding implementation details.

Lastly, I found the Real Python tutorial on Python classes and object-oriented programming to be very helpful. The whole tutorial talked about how to use classes and getter functions to get information from instances of a class. The "Encapsulation" section in particular talked about how to use getters to show information while hiding implementation details.

With the help of these resources, I learned how to use getter functions to get information from different classes and add it to my code for printing the receipt. By calling getter functions from the right classes, I was able to get the information I needed and print a receipt with all the important information about the customer and their car.

Summary of Learnings:

Through completing this assignment, which involved creating a UML diagram and writing Python code to implement the classes, I have gained the following knowledge:

1. Identifying Use Cases: I have learned how to find and write down the different ways a software system can be used. To do this, you need to know the different people involved, the tasks they need to do, and the different situations that can happen.
2. Making UML Diagrams: I have learned how to make UML diagrams, such as use-case diagrams and class diagrams. This means understanding the symbols and rules used in UML diagrams and how to use them to show the architecture of the system and how classes depend on each other.
3. How to Write Python Code: I know how to make Python classes, including their constructors, attributes, and setter/getter methods. This means knowing the syntax and structure of Python classes, as well as how to define their relationships and functions.
4. Overall, this assignment has helped me learn more about software design and development. For example, I now know how important it is to find use cases, make UML diagrams, and write Python code that works well. It's also helped me learn how to solve problems, think critically, and pay close attention to detail. Enhancing my capacity to solve problems: Developing my problem-solving skills through practice with UML diagrams and Python code serves me well in a wide variety of contexts.