# Software Modelling - UML Use Case and UML Class diagrams

Shaikha A. Shehi.

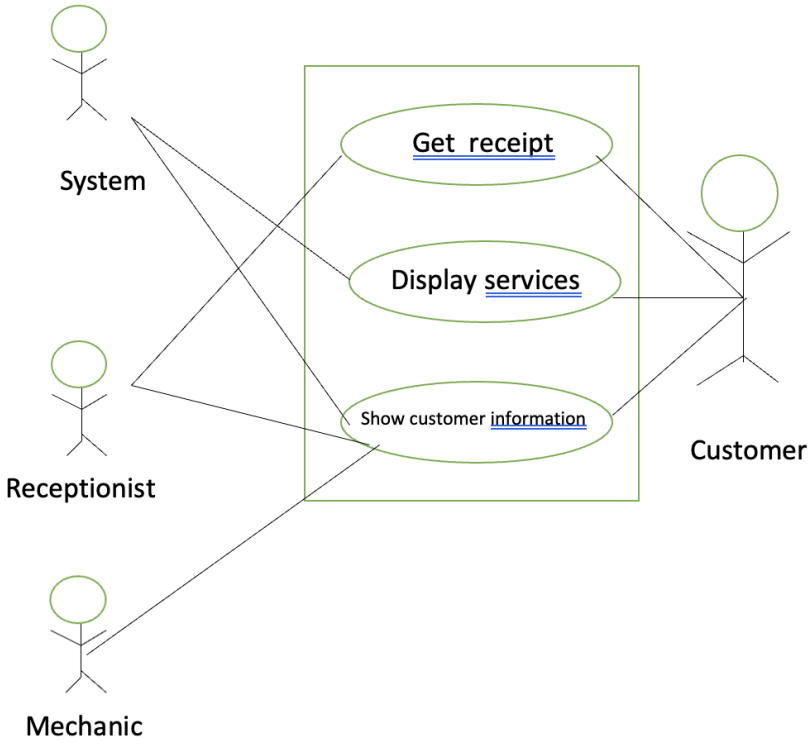
College of Interdisciplinary Studies, Zayed University

ICS 220 - programming fundamentals

Mohammed Kuhail

4th of March, 2022

UML Use-Case **Diagram and Description



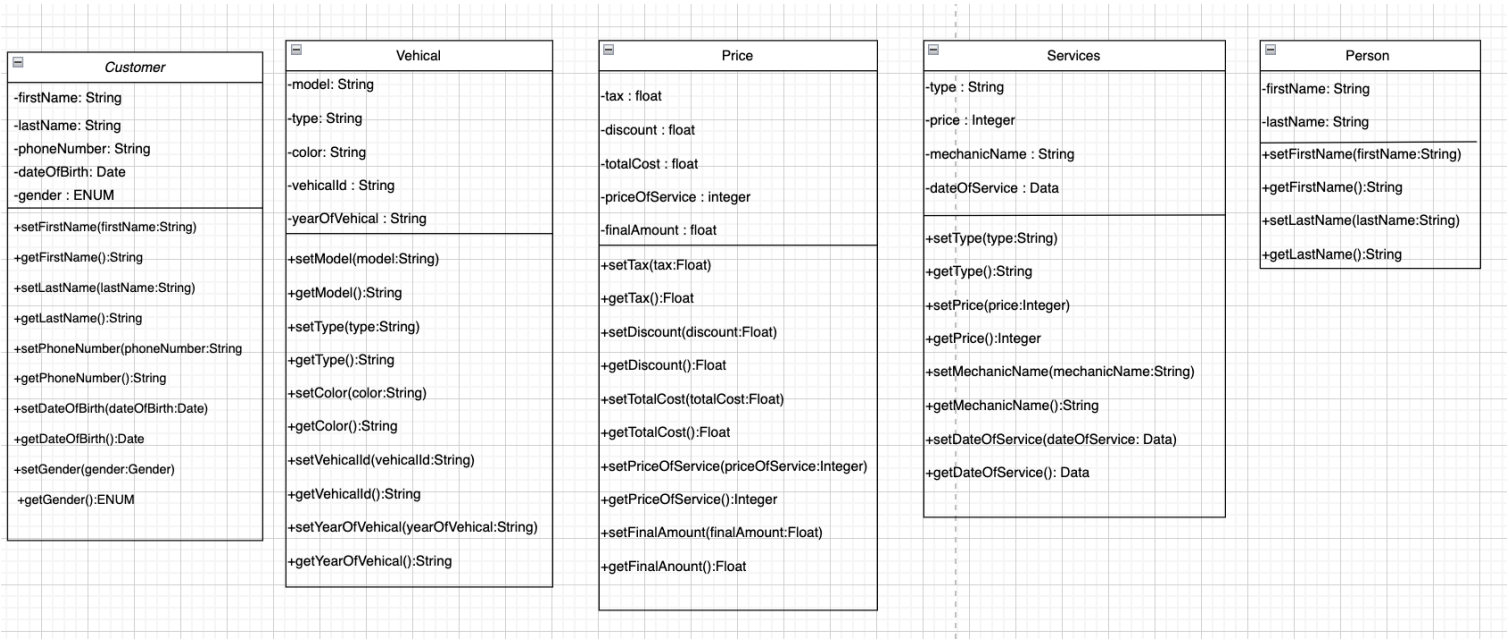

| Use case: | Get receipt |
|---|---|
| Triggers: | The user wants to get his receipt |
| Preconditions: | The user is asked for his name by the receptionist |
| Main scenarios: | |
| 1- | The user walks to the front desk and asks for his receipt |
| 2- | The receptionist asks the user for his name |
| 3- | The receptionist inputs the users name into the system |
| 4- | The system generates the services that were done |
| 5- | The system calculates the total price that the user needs to pay |

| | |
|---|---|
| 6- | The receptionist reads out the total price that the system has generated |
| 7- | The user pays the amount |
| 8- | The receipt is printed |
| Exceptions : | |
| - 3a | 1- The users name is not found within the system<br>2- The system has an eror and ask the user to reenter name |
| - 4a | 1- The system doesn't include a service that has need done to the car<br>2- The system's will generate a false receipt |
| - 5a | 1- The system miscounts the prices<br>2- The user doesn't pay full service fee |

| | |
|---|---|
| Use case: | Display services |
| Triggers : | The user wants to know the services the garage offers |
| Preconditions: | The user walks into the garage |
| Main scenarios: | |
| 1- | The user asks the receptionist for the list of services |
| 2- | The receptionist checks the system for the services |
| 3- | The system displays the list of services offered |
| 4- | The receptionist shows the list of services to the customer |
| Exceptions: | |
| - 1a | 1- There is no one at the reception desk<br>2- The customer walks away |
| - 3a | 1- The system doesn't offer a list of all the services<br>2- The system returns an error |

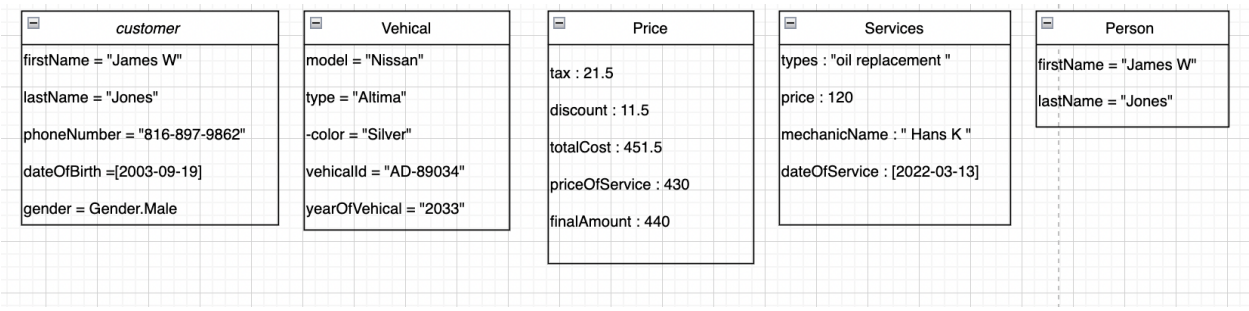
| | |
|---|---|
| Use case: | Show customer information |
| Triggers: | The mechanic wants to view the information of customer |
| Preconditions: | The mechanic log's into the system |
| Main scenarios: | |
| 1- | The mechanic clicks on the search button of the system |
| 2- | The mechanic types in the name of the customer |
| 3- | The system displays the information about the customer |
| 4- | The mechanic reads the customer's information |
| Exceptions: | |
| - 2a | 1- There customers name is not found.<br>2- The system displays an error |
| - 3a | 3- The system doesn't have any information about that customer.<br>4- The system displays the customer's page without any information |

## UML Class **Diagram and Description



In the diagram above had created class diagrams for each attribute.

UML object  **Diagram and Description




In the diagram above i created objects for each class above.

Python classes code:

```
#customer class and person using inheratance

from enum import Enum




class Gender(Enum):

   Male = "M"

   Female = "F"




class Person:
```

```
def __init__(self, first_name, last_name):

    self._first_name = first_name

    self._last_name = last_name



# setters

def set_first_name(self, first_name):

    self._first_name = first_name



def set_last_name(self, last_name):

    self._last_name = last_name



# getters

def get_first_name(self):

    return self._first_name



def get_last_name(self):

    return self._last_name
```

```
class Customer(Person):

    def __init__(self, first_name, last_name, cell_phone_number, gender, dob):

        super().__init__(first_name, last_name)

        self._cell_phone_number = cell_phone_number

        self._gender = gender

        self._dob = dob



    # setters

    def set_cell_phone_number(self, cell_phone_number):

        self._cell_phone_number = cell_phone_number



    def set_gender(self, gender):

        self._gender = gender



    def set_dob(self, dob):

        self._dob = dob
```

```
    # getters

    def get_cell_phone_number(self):

        return self._cell_phone_number



    def get_gender(self):

        return self._gender



    def get_dob(self):

        return self._dob



# a new customer object

customer = Customer("", "", "", Gender.Male, [])



# seting the attributes using the setters from both the Person and Customer
classes by inheritance

customer.set_first_name("James")

customer.set_last_name("Jones")

customer.set_cell_phone_number("816-897-9862")

customer.set_gender(Gender.Male)
```

```
customer.set_dob([2003, 9, 19])



# print the values of the attributes using the getters from both the Person and
Customer classes

print(customer.get_first_name())           # output: James

print(customer.get_last_name())            # output: Jones

print(customer.get_cell_phone_number())    # output: 816-897-9862

print(customer.get_gender())               # output: Gender.Male

print(customer.get_dob())                  # output: [2003, 9, 19]







# a new customer object

customer = Customer("", "", "", Gender.Male, [])



# seting the attributes

customer.set_first_name("James W")

customer.set_last_name("Jones")

customer.set_cell_phone_number("816-897-9862")

customer.set_dob([2003, 9, 19])
```

```
# print the values of the attributes with getters

print(customer.get_first_name())  # output: James W

print(customer.get_last_name())  # output: Jones

print(customer.get_cell_phone_number())  # output: 816-897-9862

print(customer.get_gender())  # output: Gender.Male

print(customer.get_dob())  # output: [2003, 9, 19]




#service class

class Services:

   def __init__(self, service_type, price, mechanic_name, date_of_service):

       self._service_type = service_type

       self._price = price

       self._mechanic_name = mechanic_name

       self._date_of_service = date_of_service



   # setters
```

```
def set_service_type(self, service_type):

    self._service_type = service_type



def set_price(self, price):

    self._price = price



def set_mechanic_name(self, mechanic_name):

    self._mechanic_name = mechanic_name



def set_date_of_service(self, date_of_service):

    self._date_of_service = date_of_service



# getters

def get_service_type(self):

    return self._service_type



def get_price(self):

    return self._price
```

```
    def get_mechanic_name(self):
        return self._mechanic_name

    def get_date_of_service(self):
        return self._date_of_service


# creating a new services object
services = Services("", "", "", [])


# setting the attributes
services.set_service_type("oil replacement")
services.set_price(120)
services.set_mechanic_name("Hans K")
services.set_date_of_service([2022, 3, 13])


# print the values of the attributes with getters
print(services.get_service_type())  # output: oil replacement
```

```python
print(services.get_price())          # output: 120

print(services.get_mechanic_name()) # output: Hans K

print(services.get_date_of_service())  # output: [2022, 3, 13]



#vehical class

class Vehicle:

   def __init__(self, model, vehicle_type, color, vehicle_id, year):

       self._model = model

       self._vehicle_type = vehicle_type

       self._color = color

       self._vehicle_id = vehicle_id

       self._year = year



   # setters

   def set_model(self, model):

       self._model = model



   def set_vehicle_type(self, vehicle_type):
```

```
        self._vehicle_type = vehicle_type



    def set_color(self, color):

        self._color = color



    def set_vehicle_id(self, vehicle_id):

        self._vehicle_id = vehicle_id



    def set_year(self, year):

        self._year = year



    # getters

    def get_model(self):

        return self._model



    def get_vehicle_type(self):

        return self._vehicle_type
```

```
    def get_color(self):

        return self._color



    def get_vehicle_id(self):

        return self._vehicle_id



    def get_year(self):

        return self._year



# creating a new vehicle object

vehicle = Vehicle("", "", "", "", "")



# setting the attributes

vehicle.set_model("Nissan")

vehicle.set_vehicle_type("Altima")

vehicle.set_color("Silver")

vehicle.set_vehicle_id("AD-89034")

vehicle.set_year("2033")
```

```
# print the values of the attributes with getters

print(vehicle.get_model())           # output: Nissan

print(vehicle.get_vehicle_type())    # output: Altima

print(vehicle.get_color())           # output: Silver

print(vehicle.get_vehicle_id())      # output: AD-89034

print(vehicle.get_year())            # output: 2033



#price class

class Price:

   def __init__(self, tax, total_cost, discount, final_amount, service_price):

       self._tax = tax

       self._total_cost = total_cost

       self._discount = discount

       self._final_amount = final_amount

       self._service_price = service_price



   # setters
```

```
def set_tax(self, tax):

    self._tax = tax



def set_total_cost(self, total_cost):

    self._total_cost = total_cost



def set_discount(self, discount):

    self._discount = discount



def set_final_amount(self, final_amount):

    self._final_amount = final_amount



def set_service_price(self, service_price):

    self._service_price = service_price



# getters

def get_tax(self):

    return self._tax
```

```
    def get_total_cost(self):
        return self._total_cost

    def get_discount(self):
        return self._discount

    def get_final_amount(self):
        return self._final_amount

    def get_service_price(self):
        return self._service_price
# create a new price object
price = Price(0, 0, 0, 0, 0)

# setting the attributes
price.set_tax(21.5)
price.set_discount(11.5)
```

```
price.set_total_cost(451.5)

price.set_service_price(430)

price.set_final_amount(440)



# print the values of the attributes with getters

print(price.get_tax())            # output: 21.5

print(price.get_discount())       # output: 11.5

print(price.get_total_cost())     # output: 451.5

print(price.get_service_price()) # output: 430

print(price.get_final_amount())   # output: 440
```

Github link: https://github.com/ShaikhaShehhi/Assigment-1-ICS-220

Summary of learning:

In this course so far I have gained knowledge about creating UML diagrams based on real-world entities. The skills I have gained by creating and understanding the uses of a UML diagram will hopefully be useful in the future when I need to create software with given data. Understanding the way to create a design model of any real-world entity and also using a programming language like python to help use our data and input them in the code to ease the process of retargeting the data when we need it . In this assignment, we had 5 attributes per class so it felt simple to get the code to retrieve the data however I believe if we had a larger data set then the code will be more useful in retrieving data.