Software Modelling - UML Use Case and UML Class diagrams

Table

Description automatically generated

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.
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.
3. For all the identified classes create \*\*Python classes \*\*with the constructor, attributes, and appropriate setter/getter methods. Each class must include at-least 5 attributes. Create objects of all the identified classes and use the object’s functions to populate and display the details.

Use Cases:

* Generating receipts
* Managing customer information
* Calculating taxes and discounts
* Repairing vehicles

Use Case Diagram:



Billing Management System

Cashier

Customer

Mechanic

|  |  |
| --- | --- |
| Use Case: | Generating Receipts |
| Trigger: | The costumer wants to get a receipt |
| Precondition: | The customer has received services from the auto repair shop |
| Main Scenario: |  |
| 1. | The customer makes a purchase and requests a receipt. |
| 2. | The cashier selects the "Generate Receipt" use-case. |
| 3. | The system displays a form to enter the customer information, vehicle information, and services provided. |
| 4. | The cashier enters the information and clicks "Generate Receipt" |
| 5. | The system generates a receipt and displays it on the screen |
| 6. | The cashier verifies the purchase information. |
| 7. | The cashier prints the receipt and hands it to the customer. |
| Exceptions: |  |
| 3a. | 1. The cashier enters invalid data for customer or vehicle information. 2. The system displays an error message and prompts the cashier to enter valid data. |
| 3b. | 1. The employee selects a service that is not available. 2. the system displays an error message and prompts the employee to select a valid service. |
| 5a. | 1. There is an error during the receipt generation process. 2. The system displays an error message and prompts the employee to try again. |
| 7a. | 1. There is an error during the printing process. 2. The cashier tries to print the receipt again. |

Use Case Description 1:

Use Case Description 2:

|  |  |
| --- | --- |
| Use Case: | Managing Customer Information |
| Trigger: | The auto repair shop needs to update or add customer information to maintain accurate records. |
| Precondition: | The auto repair shop has customer information that needs to be managed. |
| Main Scenario: |  |
| 1. | The employee selects the "Manage Customer Information" use-case. |
| 2. | The system displays a list of existing customers. |
| 3. | The employee selects a customer from the list or adds a new customer. |
| 4. | The system displays a form to update or add the customer information. |
| 5. | The employee updates or adds the information and clicks "Save". |
| 6. | The system saves the information and displays a confirmation message. |
| Exceptions: |  |
| 3a. | 1. There are errors during the retrieval of existing customer information. 2. the system displays an error message and prompts the employee to try again. |
| 5a. | 1. There are errors during the update or addition of customer information. 2. The system displays an error message and prompts the employee to try again. |

Use Case Description 3:

|  |  |
| --- | --- |
| Use Case: | Calculating Taxes and Discounts |
| Trigger: | The employee needs to provide the customer with an accurate invoice. |
| Precondition: | The services have been provided to the customer. |
| Main Scenario: |  |
| 1. | The customer goes to the cashier to be checked out. |
| 2. | The cashier selects the "Calculate Taxes and Discounts" use-case. |
| 3. | The system retrieves the services provided and their prices. |
| 4. | The system calculates the total cost of the services. |
| 5. | The system calculates the taxes based on the total cost and the tax rate. |
| 6. | The system applies any applicable discounts to the total cost. |
| 7. | The system displays the total cost, taxes, discounts, and the final amount to the employee. |
| Exceptions: |  |
| 3a. | 1. There are errors during the retrieval of service prices or tax rates. 2. The system displays an error message and prompts the employee to try again. |
| 4a. | 1. There are errors during the calculation of the total cost, taxes, or discounts. 2. The system displays an error message and prompts the employee to try again. |
| 7a. | 1. There are errors during the display of the total cost, taxes, discounts, and final amount. 2. The system displays an error message and prompts the employee to try again. |

Use case description 4:

|  |  |
| --- | --- |
| Use Case: | Repairing vehicles |
| Trigger: | The customer requires a vehicle repair from a mechanic. |
| Precondition: | The customer has noticed an issue with their vehicle that requires repair and seeks assistance from a mechanic. |
| Main Scenario: |  |
| 1. | The customer contacts the mechanic to request repair services for their vehicle. |
| 2. | The mechanic asks the customer to describe the problem with their vehicle and gather information about the vehicle. |
| 3. | The mechanic may ask the customer to bring the vehicle to their workshop for inspection or schedule a mobile repair service to go to the vehicle location. |
| 4. | The mechanic inspects the vehicle and diagnoses the problem. |
| 5. | The mechanic provides an estimate of the repair cost to the customer, including the parts and labor required. |
| 6. | The customer agrees to the repair cost, then the mechanic orders the necessary parts and begins the repair work. |
| 8. | Once the repair work is complete, the mechanic tests the vehicle to ensure it is in good working condition. |
| 9. | The mechanic notifies the customer that the repair work is complete and arranges for the vehicle to be returned to them or for the customer to pick it up from their workshop. |
| 10. | The customer pays for the repair services. |
| Exceptions: |  |
| 6a. | 1. The customer doesn’t agree to the repair cost. 2. The mechanic cancels the repair work. |
| 8a. | 1. The vehicle does not work. 2. The mechanic finds the issue. |
| 10a. | 1. The customer’s card gets declined. 2. The machine will send a message to try again |

The 4 scenarios:

1. James W. Jones received services from the auto repair shop, and an employee generated a receipt for him.
2. An employee added a new customer to the auto repair shop's database and updated their information.
3. An employee calculated the taxes and discounts for the services provided to a customer, and the customer received a discounted final amount.
4. The customer contacts the mechanic to request repair services for their vehicle, and the mechanic inspects and diagnoses the issue with the vehicle.Identifying classes and their objects:
5. Person:

* First Name: String
* Last Name: String
* Gender: ENUM
* Phone Number: String
* Date of Birth: Date

1. Customer:

* Email Address: String
* Billing Address: String
  + Customer1:
    - First Name: James
    - Last Name: W. Jones
    - Gender: Male
    - Phone Number: 816-897-9862
    - Date of Birth: 6-6-1996
    - Email Address: Jamesjones@gmail.com
    - Billing Address: Dubai

1. Cashier:

* Employee ID number: Integer
* Register number: Integer
* Manager name: String
  + Cashier1:
    - First Name: John
    - Last Name: Smith
    - Gender: Male
    - Employee ID number: 01234
    - Register number: 56789
    - Manager name: Tim A.

1. Mechanic:

* Areas of expertise: String
* Years of experience: Integer
  + Mechanic1:
    - First Name: Hans
    - Last Name: K
    - Gender: Male
    - Areas of expertise: Repair
    - Years of experience: 10

1. Vehicle:

* Make: ENUM
* Model: ENUM
* Year: Integer
* Color: ENUM
* Vehicle ID: String
  + Vehicle1:
    - Make: Nissan
    - Model: Altima
    - Year: 2014
    - Color: Silver
    - Vehicle ID: AD-89034

1. Service:

* Service name: ENUM
* Service time: String
* Service price: Integer
* Mechanic name: String
* Service status: ENUM
  + Service1:
    - Service name: Diagnostics
    - Service time: 1 day
    - Service price: 15
    - Mechanic name: Hans
    - Service status: Complete
  + Service2:
    - Service name: Oil replacement
    - Service time: 1 day
    - Service price: 120
    - Mechanic name: Hans
    - Service status: Complete
  + Service3:
    - Service name: Oil filter parts
    - Service time: 1 day
    - Service price: 35
    - Mechanic name: Hans
    - Service status: Complete
  + Service4:
    - Service name: Tire replacement
    - Service time: 1 day
    - Service price: 100
    - Mechanic name: Hans
    - Service status: Complete
  + Service5:
    - Service name: Tire
    - Service time: 1 day
    - Service price: 160
    - Mechanic name: Hans
    - Service status: Complete

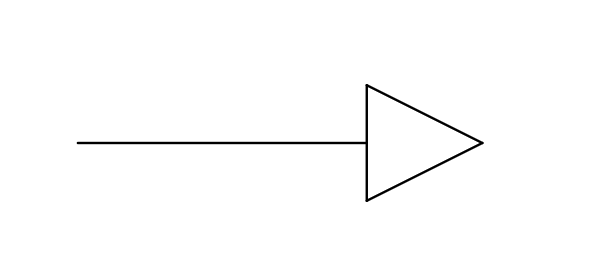
1. Receipt

* Cashier name: String
* Date: Date
* Total services: Integer
* Taxes: Float
* Discount: Float
* Total cost: Integer
  + Receipt1:
    - Cashier name: John Smith
    - Date: March 13, 2022
    - Total services: 5
    - Taxes: 21.5
    - Discount: 11.5
    - Total cost: 440

|  |
| --- |
| **Customer** |
| -emailAddress: String  -billingAddress: String |
| +getEmailAddress():String  +setEmailAddress(emailAddress:String)  +getBillingAddress():String  +setBillingAddress(billingAddress:String)  +\_str\_(): String |

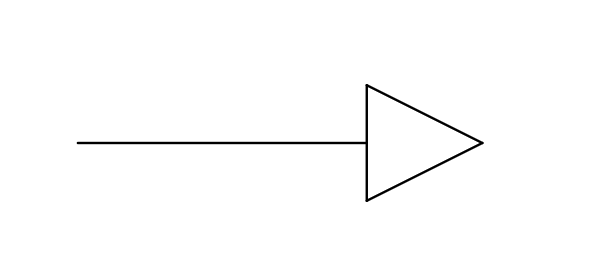
UML class diagrams:

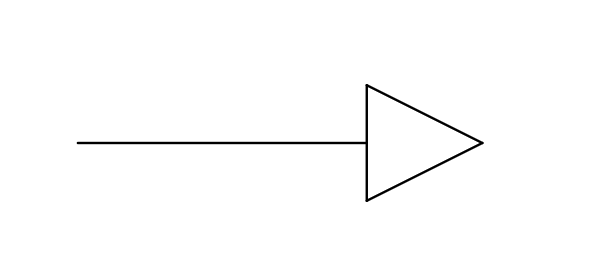
|  |
| --- |
| **Person** |
| -firstName: String  -lastName: String  -gender: ENUM  -phoneNumbr: String  -dateOfBirth: Date |
| +getFirstName():String  +setFirstName(firstName:String)  +getLastName():String  +setLastName(lastName:String)  +setGender(gender:Gender)  +getGender():ENUM  +getPhoneNumber():String  +setPhoneNumbr(phoneNumber:String)  +getDateOfBirth():Date  +setDateOfBirth (dateOfBirth:Date)  +\_str\_(): String |



|  |
| --- |
| **Cashier** |
| -employeeIDNumber: Integer  -registerNumber: Integer  -managerName: String |
| +getEmployeeIdNumber():Integer  +setEmployeeIdNumber(employeeIdNumber:Integer)  +getRegisterNumber ():Integer  +setRegisterNumber (registerNumber:Integer)  +getManagerName():String  +setManagerName(ManagerName:String)  +\_str\_(): String |

|  |
| --- |
| **Mechanic** |
| -areasOfExpertise: String  -yearsOfExperience: Integer |
| +getAreasOfExpertise():String  +setAreasOfExpertise(areasOfExpertise:String)  +getYearsOfExperience():Integer  +setYearsOfExperience(yearsOfExperience :Integer)  +\_str\_(): String |





|  |
| --- |
| **Gender** |
| male=1  female=2 |

|  |
| --- |
| **Customer1: Customer** |
| -firstName: “James”  -lastName: “W. Jones”  -gender: Gender.Male  -phoneNumbr: 816-897-9862  -dateOfBirth: [1996-6-6]  -emailAddress: “Jamesjones@gmail.com”  -billingAddress: “Dubai” |

|  |
| --- |
| **Cashier1: Cashier** |
| -firstName: “Jane”  -lastName: “Smith”  -gender: Gender.Female  -employeeIDNumber: 01234  -registerNumber: 56789  -managerName: “Tim A.” |

|  |
| --- |
| **Mechanic1: Mechanic** |
| -firstName: “Hans”  -lastName: “K”  -gender: Gender.Male  -areasOfExpertise: “Repair”  -yearsOfExperience: 10 |

|  |
| --- |
| **Make** |
| Nissan=1  Toyota=2 |

|  |
| --- |
| **Model** |
| Altima=1  Camry=2 |

|  |
| --- |
| **Color** |
| Silver=1  White=2 |

|  |
| --- |
| **Vehicle1: Vehicle** |
| -make: Make.Nissan  -model: Model.Altima  -year: 2014  -color: Color.Silver  -vehicleId: “AD-89034” |

|  |
| --- |
| **Vehicle** |
| -make: ENUM  -model: ENUM  -year: Integer  -color: ENUM  -vehicleId: String |
| +getMake():ENUM  +setMake(make:ENUM)  +getModel():ENUM  +setModel(model:ENUM)  +getYear():Integer  +setYear(year:Integer)  +getColor():ENUM  +setColor(color:ENUM)  +getVehicleId():String  +setVehicleId(vehicleId:String)  +\_str\_(): String |

|  |
| --- |
| **Service** |
| -serviceName: ENUM  -servicePrice: Integer  -mechanicName: String  -serviceStatus: ENUM |
| +getServiceName():ENUM  +setServiceName(serviceName:ENUM)  +getServicePrice():Integer  +setServicePrice(servicePrice:Integer)  +getMechanicName():String  +setMechanicName(mechanicName:String)  +getServiceStatus():ENUM  +setServiceStatus(serviceStatus :ENUM)  +\_str\_(): String |

|  |
| --- |
| **ServiceName** |
| Diagnostics=1  OilReplacement=2  OilFilterParts=3  TireReplacement=4  Tire=5 |

|  |
| --- |
| **ServiceStatus** |
| Pending=1  In progress=2  Complete=3 |

|  |
| --- |
| **Service1: Service** |
| -serviceName: ServiceName.Diagnostics  -serviceTime: 1 day  -servicePrice: 15  -mechanicName: “Hans K.”  -serviceStatus: ServiceStatus.Complete |

|  |
| --- |
| **Service2: Service** |
| -serviceName: ServiceName.OilReplacement  -serviceTime: 1 day  -servicePrice: 120  -mechanicName: “Hans K.”  -serviceStatus: ServiceStatus.Complete |

|  |
| --- |
| **Service3: Service** |
| -serviceName: ServiceName.OilFilterParts  -serviceTime: 1 day  -servicePrice: 35  -mechanicName: “Hans K.”  -serviceStatus: ServiceStatus.Complete |

|  |
| --- |
| **Service4: Service** |
| -serviceName: ServiceName.TireReplacement  -serviceTime: 1 day  -servicePrice: 100  -mechanicName: “Hans K.”  -serviceStatus: ServiceStatus.Complete |

|  |
| --- |
| **Service5: Service** |
| -serviceName: ServiceName.Tire  -serviceTime: 1 day  -servicePrice: 160  -mechanicName: “Hans K.”  -serviceStatus: ServiceStatus.Complete |

|  |
| --- |
| **Receipt** |
| cashierName: String  date: Date  totalServices: Integer  taxes: Float  discount: Float  totalCost: Integer |
| +getCashierName():String  +setCashierName(cashierName:String)  +getDate():Date  +setDate(date:Date)  +getTotalServices():Integer  +setTotalServices(totalServices:Integer)  +getTaxes():Float  +setTaxes(taxes:Float)  +getDiscount():Float  +setDiscount(discount:Float)  +getTotalCost():Integer  +setTotalCost(totalCost:Integer)  +\_str\_(): String |

|  |
| --- |
| **Receipt1: Receipt** |
| -cashierName: “John Smith”  -date: [3- 13-2022]  -totalServices: 5  -taxes: 21.5  -discount: 11.5  -totalCost: 440 |

Relationships:

Person is a superclass of Customer, Cashier, and Mechanic classes. Each of these classes has additional attributes and methods specific to their roles.

Customer is a Person

Cashier is a Person

Mechanic is a Person

Vehicle is associated with Service as each service is performed on a specific vehicle.

The Employee class is associated with the Receipt class through a dependency relationship, as an employee interacts with the system to generate a receipt.

* A Customer can have one or more Receipts associated with them. This is a one-to-many relationship, where the Customer class is the "one" side and the Receipt class is the "many" side.
* A Cashier can have one or more Receipts associated with them. This is also a one-to-many relationship, where the Cashier class is the "one" side and the Receipt class is the "many" side.
* A Mechanic can perform one or more Services. This is a one-to-many relationship, where the Mechanic class is the "one" side and the Service class is the "many" side.
* A Service can be performed by one Mechanic and can be associated with one Vehicle. This is a many-to-one relationship, where the Mechanic and Vehicle classes are the "many" side, and the Service class is the "one" side.
* A Receipt can contain one or more Services. This is a one-to-many relationship, where the Receipt class is the "one" side and the Service class is the "many" side.
* A Vehicle can have one or more Services associated with it. This is a one-to-many relationship, where the Vehicle class is the "one" side and the Service class is the "many" side.
* Each Service can be associated with only one Vehicle. This is a many-to-one relationship, where the Vehicle class is the "many" side and the Service class is the "one" side.
* Each Service can be associated with only one Receipt. This is also a many-to-one relationship, where the Receipt class is the "many" side and the Service class is the "one" side.
* Each Receipt can have only one Customer associated with it. This is a many-to-one relationship, where the Customer class is the "many" side and the Receipt class is the "one" side.

Python:

# Enums  
from enum import Enum  
class Gender(Enum):  
 male = 1  
 female = 2  
class Make(Enum):  
 Nissan = 1  
 Toyota = 2  
class Model(Enum):  
 Altima = 1  
 Camry = 2  
class Color(Enum):  
 Silver=1  
 White=2  
class ServiceName(Enum):  
 Diagnostics = 1  
 OilReplacement = 2  
 OilFilterParts = 3  
 TireReplacement = 4  
 Tire = 5  
class ServiceStatus(Enum):  
 Pending = 1  
 Inprogress = 2  
 Complete = 3  
  
  
class Person:  
 def \_\_init\_\_(self, firstName, lastName, gender, phoneNumber, dateOfBirth):  
 self.\_\_firstName = firstName  
 self.\_\_lastName = lastName  
 self.\_\_gender = gender  
 self.\_\_phoneNumber = phoneNumber  
 self.\_\_dateOfBirth = dateOfBirth  
  
 def getFirstName(self):  
 return self.\_\_firstName  
  
 def getLastName(self):  
 return self.\_\_lastName  
  
 def getGender(self):  
 return self.\_\_gender  
  
 def getPhoneNumber(self):  
 return self.\_\_phoneNumber  
  
 def getDateOfBirth(self):  
 return self.\_\_dateOfBirth  
  
 def setFirstName(self, firstName):  
 self.\_\_firstName = firstName  
  
 def setLastName(self, lastName):  
 self.\_\_lastName = lastName  
  
 def setGender(self, gender):  
 self.\_\_gender = gender  
  
 def setPhoneNumber(self, phoneNumber):  
 self.\_\_phoneNumber = phoneNumber  
  
 def setDateOfBirth(self, dateOfBirth):  
 self.\_\_dateOfBirth = dateOfBirth  
  
 def \_\_str\_\_(self):  
 return f"Name: {self.\_\_firstName} {self.\_\_lastName}, Gender: {self.\_\_gender}, Phone: {self.\_\_phoneNumber}, DOB: {self.\_\_dateOfBirth}"  
  
  
class Customer(Person):  
 def \_\_init\_\_(self, firstName, lastName, gender, phoneNumber, dateOfBirthirth, email, billingAddress):  
 super().\_\_init\_\_(firstName, lastName, gender, phoneNumber, dateOfBirthirth)  
 self.\_\_email = email  
 self.\_\_billingAddress = billingAddress  
  
 def getEmail(self):  
 return self.\_\_email  
  
 def getBillingAddress(self):  
 return self.\_\_billingAddress  
  
 def setEmail(self, email):  
 self.\_\_email = email  
  
 def setBillingAddress(self, billingAddress):  
 self.\_\_billingAddress = billingAddress  
  
 def \_\_str\_\_(self):  
 return f"{super().\_\_str\_\_()}, Email: {self.\_\_email}, Billing Address: {self.\_\_billingAddress}"  
  
class Cashier(Person):  
 def \_\_init\_\_(self, firstName, lastName, gender, employeeIdNumber, registerNumber, managerName):  
 super().\_\_init\_\_(firstName, lastName, gender, "",  
 "") # empty string values for phone\_number and date\_of\_birth  
 self.\_\_employeeIdNumber = employeeIdNumber  
 self.\_\_registerNumber = registerNumber  
 self.\_\_managerName = managerName  
  
 def getEmployeeIdNumber(self):  
 return self.\_\_employeeIdNumber  
  
 def getRegisterNumber(self):  
 return self.\_\_registerNumber  
  
 def getManagerName(self):  
 return self.\_\_managerName  
  
 def setEmployeeIdNumber(self, employeeIdNumber):  
 self.\_\_employeeIdNumber = employeeIdNumber  
  
 def setRegisterNumber(self, registerNumber):  
 self.\_\_registerNumber = registerNumber  
  
 def setManagerName(self, managerName):  
 self.\_\_managerName = managerName  
  
 def \_\_str\_\_(self):  
 return f"Name: {self.getFirstName()} {self.getLastName()}, Gender: {self.getGender()}, Employee ID: {self.\_\_employeeIdNumber}, Register: {self.\_\_registerNumber}, Manager: {self.\_\_managerName}"  
  
  
class Mechanic(Person):  
 def \_\_init\_\_(self, firstName, lastName, gender, areasOfExpertise, yearsOfExperience):  
 super().\_\_init\_\_(firstName, lastName, gender, "", "")  
 self.\_\_areasOfExpertise = areasOfExpertise  
 self.\_\_yearsOfExperience = yearsOfExperience  
  
 def getAreasOfExpertise(self):  
 return self.\_\_areasOfExpertise  
  
 def setAreasOfExpertise(self, areasOfExpertise):  
 self.\_\_areasOfExpertise = areasOfExpertise  
  
 def getYearsOfExperience(self):  
 return self.\_\_yearsOfExperience  
  
 def setYearsOfExperience(self, yearsOfExperience):  
 self.\_\_yearsOfExperience = yearsOfExperience  
  
 def \_\_str\_\_(self):  
 return f"Name: {self.getFirstName()} {self.getLastName()}, Gender: {self.getGender()}, Areas of Expertise: {self.\_\_areasOfExpertise}, Years of Experience: {self.\_\_yearsOfExperience}"  
  
  
class Vehicle:  
 def \_\_init\_\_(self, make, model, year, color, vehicleId):  
 self.\_\_make = make  
 self.\_\_model = model  
 self.\_\_year = year  
 self.\_\_color = color  
 self.\_\_vehicleId = vehicleId  
  
 def getMake(self):  
 return self.\_\_make  
  
 def set\_make(self, make):  
 self.\_\_make = make  
  
 def getModel(self):  
 return self.\_\_model  
  
 def setModel(self, model):  
 self.\_\_model = model  
  
 def getYear(self):  
 return self.\_\_year  
  
 def setYear(self, year):  
 self.\_\_year = year  
  
 def getColor(self):  
 return self.\_\_color  
  
 def setColor(self, color):  
 self.\_\_color = color  
  
 def getVehicleId(self):  
 return self.\_\_vehicleId  
  
 def setVehicleId(self, vehicleId):  
 self.\_\_vehicleId = vehicleId  
  
 def \_\_str\_\_(self):  
 return f"Make: {self.\_\_make}, Model: {self.\_\_model}, Year: {self.\_\_year}, Color: {self.\_\_color}, Vehicle ID: {self.\_\_vehicleId}"  
  
class Service():  
 def \_\_init\_\_(self, serviceName, serviceTime, servicePrice, mechanicName, serviceStatus):  
 self.\_\_serviceName = serviceName  
 self.\_\_serviceTime = serviceTime  
 self.\_\_servicePrice = servicePrice  
 self.\_\_mechanicName = mechanicName  
 self.\_\_serviceStatus = serviceStatus  
  
 def setServiceName(self, serviceName):  
 self.\_\_serviceName = serviceName  
  
 def getServiceName(self):  
 return self.\_\_serviceName  
  
 def setServicePrice(self, servicePrice):  
 self.\_\_servicePrice = servicePrice  
  
 def getServicePrice(self):  
 return self.\_\_servicePrice  
  
 def setMechanicName(self, mechanicName):  
 self.\_\_mechanicName = mechanicName  
  
 def getMechanicName(self):  
 return self.\_\_mechanicName  
  
 def setServiceStatus(self, serviceStatus):  
 self.\_\_serviceStatus = serviceStatus  
  
 def getServiceStatus(self):  
 return self.\_\_serviceStatus  
  
 def \_\_str\_\_(self):  
 return f"Service Name: {self.\_\_serviceName}, service price: {self.\_\_servicePrice}, mechanic name: {self.\_\_mechanicName}, service status: {self.\_\_serviceStatus}"  
  
class Receipt:  
 def \_\_init\_\_(self, cashierName, date, totalServices, taxes, discount, totalCost):  
 self.\_\_cashierName = cashierName  
 self.\_\_date = date  
 self.\_\_totalServices = totalServices  
 self.\_\_taxes = taxes  
 self.\_\_discount = discount  
 self.\_\_totalCost = totalCost  
  
 def setCashierName(self, name):  
 self.\_\_cashierName = name  
  
 def getCashierName(self):  
 return self.\_\_cashierName  
  
 def setDate(self, date):  
 self.\_\_date = date  
  
 def getDate(self):  
 return self.\_\_date  
  
 def setTotalServices(self, totalServices):  
 self.\_\_totalServices = totalServices  
  
 def getTotalServices(self):  
 return self.\_\_totalServices  
  
 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 setTotalCost(self, totalCost):  
 self.\_\_totalCost = totalCost  
  
 def getTotalCost(self):  
 return self.\_\_totalCost  
  
 def \_\_str\_\_(self):  
 return f"Cashier: {self.\_\_cashierName}\nDate: {self.\_\_date}\nTotal Services: {self.\_\_totalServices}\nTaxes: {self.\_\_taxes}\nDiscount: {self.\_\_discount}\nTotal Cost: {self.\_\_totalCost}"  
  
# Create a Customer object  
customer1 = Customer("James", "W. Jones", Gender.male.name, "816-897-9862", "1996-06-06", "Jamesjones@gmail.com", "Dubai")  
print(customer1)  
  
# Create a Cashier object  
cashier1 = Cashier("Jane", "Smith", Gender.female.name, "01234", "56789" , "Tim A.")  
print(cashier1)  
  
# Create a Mechanic object  
mechanic1 = Mechanic("Hans", "K", Gender.male.name, "Repair", 10)  
print(mechanic1)  
  
# Create a Vehicle object  
vehicle1 = Vehicle(Make.Nissan.name, Model.Altima.name, 2010, Color.Silver.name, "AD-89034")  
print(vehicle1)  
  
# Create a Service object  
service1 = Service(ServiceName.Diagnostics.name, "1 hour", 15, "Hans K.", "Completed")  
service2 = Service(ServiceName.OilReplacement.name, "1 hour", 120, "Hans K.", "Completed")  
service3 = Service(ServiceName.OilFilterParts.name, "1 hour", 35, "Hans K.", "Completed")  
service4 = Service(ServiceName.TireReplacement.name, "1 hour", 100, "Hans K.", "Completed")  
service5 = Service(ServiceName.Tire.name, "1 hour", 160, "Hans K.", "Completed")  
  
  
  
print(service1)  
print(service2)  
print(service3)  
print(service4)  
print(service5)  
  
  
  
# Create a Receipt object  
receipt = Receipt("John Smith", "2022-03-13", 5, 21.5, 11.5, 440)  
print(receipt)

Output:

Name: James W. Jones, Gender: male, Phone: 816-897-9862, DOB: 1996-06-06, Email: Jamesjones@gmail.com, Billing Address: Dubai

Name: Jane Smith, Gender: female, Employee ID: 01234, Register: 56789, Manager: Tim A.

Name: Hans K, Gender: male, Areas of Expertise: Repair, Years of Experience: 10

Make: Nissan, Model: Altima, Year: 2010, Color: Silver, Vehicle ID: AD-89034

Service Name: Diagnostics, service price: 15, mechanic name: Hans K., service status: Completed

Service Name: OilReplacement, service price: 120, mechanic name: Hans K., service status: Completed

Service Name: OilFilterParts, service price: 35, mechanic name: Hans K., service status: Completed

Service Name: TireReplacement, service price: 100, mechanic name: Hans K., service status: Completed

Service Name: Tire, service price: 160, mechanic name: Hans K., service status: Completed

Cashier: John Smith

Date: 2022-03-13

Total Services: 5

Taxes: 21.5

Discount: 11.5

Total Cost: 440

Summary:

The task of creating a billing system for a vehicle repair service allowed me to apply my knowledge of object-oriented programming that I have learned in class to a real-world scenario. I found this assignment both challenging and rewarding. I had to brainstorm and experiment with different ideas for implementing the billing system, considering various use cases and scenarios that could arise. Throughout the assignment, I made several drafts, constantly refining my work to make it more precise and structured. I learned how to create classes with attributes and instantiate them, and how inheritance can simplify the code and make it more manageable. Overall, learning about UML use case and class diagrams has been crucial in helping me understand how to design and build object-oriented systems. By having a clear understanding of the use cases and relationships between objects, I can create efficient and effective systems that meet the needs of their users.

* I have created a use-case diagram to represent the different actions and roles involved in a typical garage service.
* I created a class diagram to represent the relationships between different objects involved in the garage service system. The classes included Customer, Vehicle, Service, Cashier, and Receipt.
* I implemented Python classes for each of the identified classes, including their constructor, attributes, and appropriate getter and setter methods.
* I created objects of the identified classes and used their functions to display their information.

Github Link:

<https://github.com/alyalootah/Assignment1>