Rushalee Das

Car Rental System

Create following tables in SQL Schema with appropriate class and write the unit test case for the Car Rental application.

Schema Design:

1. **Vehicle Table:**
    - vehicleID (Primary Key)
    - make
    - model
    - year
    - dailyRate
    - status (available, notAvailable)
    - passengerCapacity
    - engineCapacity
2. **Customer Table:**
    - customerID (Primary Key)
    - firstName
    - lastName
    - email • phoneNumber
3. **Lease Table:**
    - leaseID (Primary Key)
    - vehicleID (Foreign Key referencing Vehicle Table)
    - customerID (Foreign Key referencing Customer Table)
    - startDate
    - endDate
    - type (to distinguish between DailyLease and MonthlyLease)
4. **Payment Table:**
    - paymentID (Primary Key)
    - leaseID (Foreign Key referencing Lease Table)
    - paymentDate
    - amount

```
mysql> CREATE DATABASE CarRentalSystem;
Query OK, 1 row affected (0.02 sec)
```

```
mysql> USE CarRentalSystem;
Database changed
mysql> CREATE TABLE Vehicle (
    ->     vehicleID INT PRIMARY KEY,
    ->     make VARCHAR(255),
    ->     model VARCHAR(255),
    ->     year INT,
    ->     dailyRate DECIMAL(10, 2),
    ->     status ENUM('available', 'notAvailable'),
    ->     passengerCapacity INT,
    ->     engineCapacity INT
    -> );
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> CREATE TABLE Customer (
    ->     customerID INT PRIMARY KEY,
    ->     firstName VARCHAR(255),
    ->     lastName VARCHAR(255),
    ->     email VARCHAR(255),
    ->     phoneNumber VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> CREATE TABLE Lease (
    ->     leaseID INT PRIMARY KEY,
    ->     vehicleID INT,
    ->     customerID INT,
    ->     startDate DATE,
    ->     endDate DATE,
    ->     type ENUM('DailyLease', 'MonthlyLease'),
    ->     FOREIGN KEY (vehicleID) REFERENCES Vehicle(vehicleID),
    ->     FOREIGN KEY (customerID) REFERENCES Customer(customerID)
    -> );
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> CREATE TABLE Payment (
    ->     paymentID INT PRIMARY KEY,
    ->     leaseID INT,
    ->     paymentDate DATE,
    ->     amount DECIMAL(10, 2),
    ->     FOREIGN KEY (leaseID) REFERENCES Lease(leaseID)
    -> );
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> DESC Vehicle;
+-------------------+---------------------------------+------+-----+---------+-------+
| Field             | Type                            | Null | Key | Default | Extra |
+-------------------+---------------------------------+------+-----+---------+-------+
| vehicleID         | int                             | NO   | PRI | NULL    |       |
| make              | varchar(255)                    | YES  |     | NULL    |       |
| model             | varchar(255)                    | YES  |     | NULL    |       |
| year              | int                             | YES  |     | NULL    |       |
| dailyRate         | decimal(10,2)                   | YES  |     | NULL    |       |
| status            | enum('available','notAvailable') | YES  |     | NULL    |       |
| passengerCapacity | int                             | YES  |     | NULL    |       |
| engineCapacity    | int                             | YES  |     | NULL    |       |
+-------------------+---------------------------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

```
mysql> DESC Customer;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| customerID  | int          | NO   | PRI | NULL    |       |
| firstName   | varchar(255) | YES  |     | NULL    |       |
| lastName    | varchar(255) | YES  |     | NULL    |       |
| email       | varchar(255) | YES  |     | NULL    |       |
| phoneNumber | varchar(20)  | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> DESC Lease;
+------------+-----------------------------------+------+-----+---------+-------+
| Field      | Type                              | Null | Key | Default | Extra |
+------------+-----------------------------------+------+-----+---------+-------+
| leaseID    | int                               | NO   | PRI | NULL    |       |
| vehicleID  | int                               | YES  | MUL | NULL    |       |
| customerID | int                               | YES  | MUL | NULL    |       |
| startDate  | date                              | YES  |     | NULL    |       |
| endDate    | date                              | YES  |     | NULL    |       |
| type       | enum('DailyLease','MonthlyLease') | YES  |     | NULL    |       |
+------------+-----------------------------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
mysql> DESC Payment;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| paymentID   | int           | NO   | PRI | NULL    |       |
| leaseID     | int           | YES  | MUL | NULL    |       |
| paymentDate | date          | YES  |     | NULL    |       |
| amount      | decimal(10,2) | YES  |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

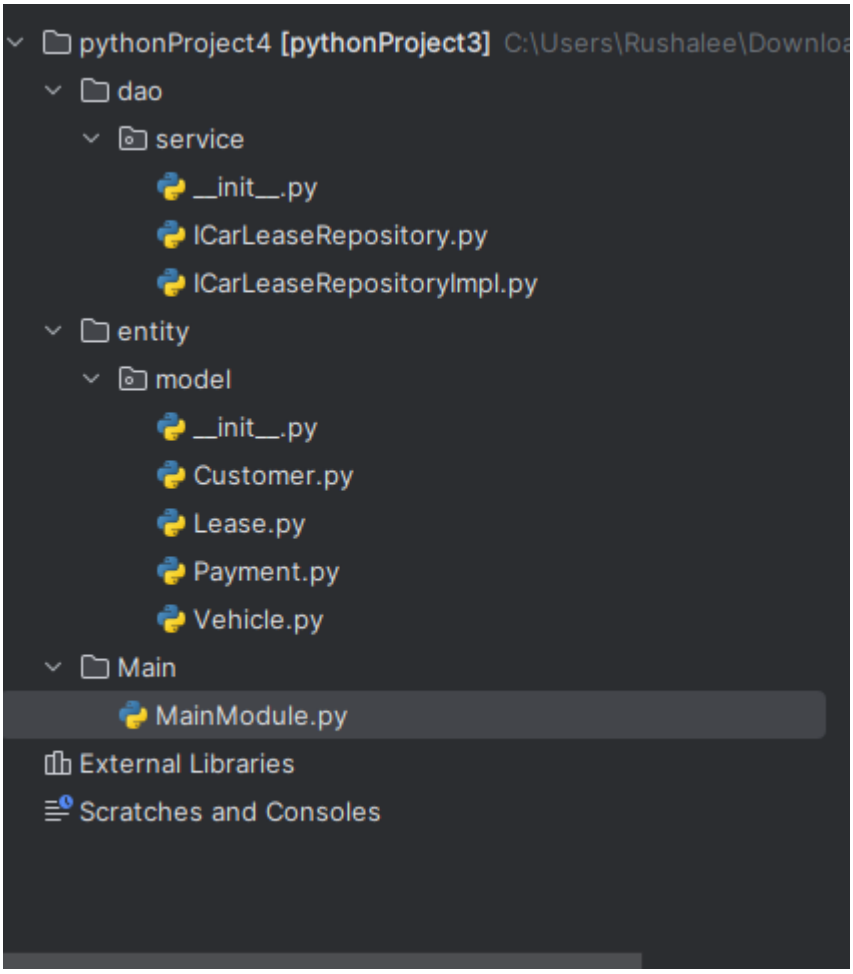
```
mysql> SELECT * FROM Vehicle;
+-----------+--------+--------+------+-----------+-----------+------------------+----------------+
| vehicleID | make   | model  | year | dailyRate | status    | passengerCapacity | engineCapacity |
+-----------+--------+--------+------+-----------+-----------+------------------+----------------+
|         1 | Toyota | Camry  | 2019 |     50.00 | available |                5 |           2000 |
|         2 | Honda  | Accord | 2020 |     55.00 | available |                5 |           2200 |
|         3 | Ford   | Fusion | 2018 |     45.00 | available |                5 |           1800 |
+-----------+--------+--------+------+-----------+-----------+------------------+----------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Customer;
+------------+-----------+----------+---------------------------+-------------+
| customerID | firstName | lastName | email                     | phoneNumber |
+------------+-----------+----------+---------------------------+-------------+
|          1 | John      | Doe      | john.doe@example.com      | 1234567890  |
|          2 | Jane      | Smith    | jane.smith@example.com    | 9876543210  |
|          3 | Michael   | Johnson  | michael.johnson@example.com | 5555555555  |
+------------+-----------+----------+---------------------------+-------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Lease;
+---------+-----------+------------+------------+------------+--------------+
| leaseID | vehicleID | customerID | startDate  | endDate    | type         |
+---------+-----------+------------+------------+------------+--------------+
|       1 |         1 |          1 | 2024-02-01 | 2024-02-06 | DailyLease   |
|       2 |         2 |          2 | 2024-02-02 | 2024-02-28 | MonthlyLease |
|       3 |         3 |          3 | 2024-02-03 | 2024-02-08 | DailyLease   |
+---------+-----------+------------+------------+------------+--------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Payment;
+-----------+---------+-------------+---------+
| paymentID | leaseID | paymentDate | amount  |
+-----------+---------+-------------+---------+
|         1 |       1 | 2024-02-06  |  250.00 |
|         2 |       2 | 2024-02-28  | 1650.00 |
|         3 |       3 | 2024-02-08  |  225.00 |
+-----------+---------+-------------+---------+
3 rows in set (0.00 sec)
```

5. **Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)**



6. **Service Provider Interface/Abstract class:**
   Keep the interfaces and implementation classes in package dao
   - Create Interface for **ICarLeaseRepository** and add following methods which interact with database.
   - **Car Management**
     1. **addCar(Car car)** parameter : Car
        return type : void
     2. **removeCar()** parameter : carID
        return type : void
     3. listAvailableCars() - parameter: NIL
        return type: return List of Car
     4. listRentedCars() – return List of Car
        parameter: NIL
        return type: return List of Car

5. findCarById(int carID) – return Car if found or throw exception
   parameter: NIL
   return type: return List of Car

- **Customer Management**
  1. addCustomer(Customer customer) parameter : Customer
     return type : void
  2. void removeCustomer(int customerID) parameter : CustomerID return type : void
  3. listCustomers() parameter : NIL return type : list of customer
  4. findCustomerById(int customerID) parameter : CustomerID
     return type : Customer
- **Lease Management**
  1. createLease() parameter : int customerID, int carID, Date startDate, Date endDate return type : Lease
  2. void returnCar(); parameter : int leaseID return type : Lease info
  3. List<Lease> listActiveLeases(); parameter : NIL return type : Lease list
  4. listLeaseHistory(); parameter : NIL
     return type : Lease list
- **Payment Handling**
  1. void recordPayment(); parameter : Lease lease, double amount return type : void

```
class Customer:
    def __init__(self, customerID, firstName, lastName, email, phoneNumber):
        self.customerID = customerID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.phoneNumber = phoneNumber
```

```
class Lease:
    def __init__(self, leaseID, vehicleID, customerID, startDate, endDate):
```

```
        self.leaseID = leaseID
        self.vehicleID = vehicleID
        self.customerID = customerID
        self.startDate = startDate
        self.endDate = endDate
```

```
class Payment:
    def __init__(self, paymentID, leaseID, paymentDate, amount):
        self.paymentID = paymentID
        self.leaseID = leaseID
        self.paymentDate = paymentDate
        self.amount = amount
```

```
class Vehicle:
    def __init__(self, vehicleID, make, model, year, dailyRate, status,
passengerCapacity, engineCapacity):
        self.vehicleID = vehicleID
        self.make = make
        self.model = model
        self.year = year
        self.dailyRate = dailyRate
        self.status = status
        self.passengerCapacity = passengerCapacity
        self.engineCapacity = engineCapacity
```

7. Implement the above interface in a class called **ICarLeaseRepositoryImpl in package dao**.

```
from abc import ABC, abstractmethod
from datetime import date
class ICarLeaseRepository(ABC):
    @abstractmethod
    def addCar(self, car) :
        pass

    @abstractmethod
    def removeCar(self):
        pass

    @abstractmethod
    def listAvailableCars(self) :
        pass

    @abstractmethod
    def listRentedCars(self):
        pass

    @abstractmethod
    def findCarById(self, carID):
        pass

    @abstractmethod
    def addCustomer(self, customer):
        pass
```

```
    @abstractmethod
    def removeCustomer(self, customerID):
        pass

    @abstractmethod
    def listCustomers(self) :
        pass

    @abstractmethod
    def findCustomerById(self, customerID):
        pass

    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate: date,
endDate: date) :
        pass

    @abstractmethod
    def returnCar(self, leaseID: int):
        pass

    @abstractmethod
    def listActiveLeases(self):
        pass

    @abstractmethod
    def listLeaseHistory(self) :
        pass

    @abstractmethod
    def recordPayment(self, lease, amount: float):
        pass
```

Connect your application to the SQL database:

8. Connect your application to the SQL database and write code to establish a connection to your SQL database.
    - Create a utility class **DBConnection** in a package **util** with a static variable **connection** of Type **Connection** and a static method **getConnection()** which returns connection.
    - Connection properties supplied in the connection string should be read from a property file.
    - Create a utility class **PropertyUtil** which contains a static method named **getPropertyString()** which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```
import mysql.connector
from mysql.connector import Error

from dao.service.ICarLeaseRepository import ICarLeaseRepository
from entity.model.Lease import Lease
```

```
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self, connection_params):
        self.connection_params = connection_params
        self.connection = self.create_connection()

    def create_connection(self):
        try:
            connection =
mysql.connector.connect(**self.connection_params)
            if connection.is_connected():
                print("Connected to MySQL database")
                return connection
        except Error as e:
            print(f"Error: {e}")
            return None

    def close_connection(self):
        if self.connection.is_connected():
            self.connection.close()
            print("Connection closed")


    def addCar(self, car):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO vehicle (vehicleID, make, model, year,
dailyRate, status, passengerCapacity, engineCapacity) VALUES (%s, %s,
%s, %s, %s, %s, %s, %s)"
            values = (car.vehicleID, car.make, car.model, car.year,
car.dailyRate, car.status, car.passengerCapacity, car.engineCapacity)
            cursor.execute(query, values)
            self.connection.commit()
            print("Car added successfully")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def removeCar(self, carID) :
        try:
            cursor = self.connection.cursor()
            query = "DELETE FROM vehicle WHERE vehicleID = %s"
            values = (carID,)
            cursor.execute(query, values)
            self.connection.commit()
            print("Car removed successfully")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def listAvailableCars(self):
        try:
            cursor = self.connection.cursor()
```

```
            query = "SELECT * FROM vehicle WHERE status = 'available'"
            cursor.execute(query)
            result = cursor.fetchall()
            return result
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def listRentedCars(self) :
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM vehicle WHERE status =
'notAvailable'"
            cursor.execute(query)
            result = cursor.fetchall()

            return result
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def findCarById(self, carID):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM vehicle WHERE vehicleID = %s"
            values = (carID,)
            cursor.execute(query, values)
            result = cursor.fetchone()
            if result:
                return result
            else:
                raise Exception(f"Car with ID {carID} not found")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def addCustomer(self, customer):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO customer (customerID, firstName,
lastName, email, phoneNumber) VALUES (%s, %s, %s, %s, %s)"
            values = (customer.customerID, customer.firstName,
customer.lastName, customer.email, customer.phoneNumber)
            cursor.execute(query, values)
            self.connection.commit()
            print("Customer added successfully")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def removeCustomer(self, customerID: int) :
        try:
```

```
            cursor = self.connection.cursor()
            query = "DELETE FROM customer WHERE customerID = %s"
            values = (customerID,)
            cursor.execute(query, values)
            self.connection.commit()
            print("Customer removed successfully")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def listCustomers(self) :
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM customer"
            cursor.execute(query)
            result = cursor.fetchall()
            return result
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def findCustomerById(self, customerID: int):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM customer WHERE customerID = %s"
            values = (customerID,)
            cursor.execute(query, values)
            result = cursor.fetchone()
            if result:
                return result
            else:
                raise Exception(f"Customer with ID {customerID} not
found")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def createLease(self, customerID: int, carID: int, startDate,
endDate):
        try:
            cursor = self.connection.cursor()

            # Fetch the current maximum leaseID from the database
            cursor.execute("SELECT MAX(leaseID) FROM lease")
            max_lease_id = cursor.fetchone()[0]

            # Increment the max_lease_id by 1 to get the new leaseID
            new_lease_id = max_lease_id + 1 if max_lease_id is not None
else 1

            # Insert the new lease record with the calculated leaseID
            query = "INSERT INTO lease (leaseID, vehicleID, customerID,
startDate, endDate) VALUES (%s, %s, %s, %s, %s)"
```

```
            values = (new_lease_id, carID, customerID, startDate,
endDate)
            cursor.execute(query, values)
            self.connection.commit()

            # Return the Lease object with the calculated leaseID
            return Lease(new_lease_id, carID, customerID, startDate,
endDate)

        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def returnCar(self, leaseID: int):
        try:
            cursor = self.connection.cursor()
            query = "UPDATE lease SET endDate = CURRENT_DATE WHERE
leaseID = %s"
            values = (leaseID,)
            cursor.execute(query, values)
            self.connection.commit()

            # Fetch the updated lease information
            query_select = "SELECT * FROM lease WHERE leaseID = %s"
            cursor.execute(query_select, values)
            result = cursor.fetchone()
            if result:
                return result
            else:
                raise Exception(f"Lease with ID {leaseID} not found")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def listActiveLeases(self):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM lease WHERE endDate >= CURRENT_DATE"
            cursor.execute(query)
            result = cursor.fetchall()
            return result
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def listLeaseHistory(self):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM lease WHERE endDate < CURRENT_DATE"
            cursor.execute(query)
            result = cursor.fetchall()

            return result
```

```
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def recordPayment(self, leaseID: int, amount: float):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO payment (leaseID, paymentDate, amount)
VALUES (%s, CURRENT_DATE, %s)"
            values = (leaseID, amount)
            cursor.execute(query, values)
            self.connection.commit()
            print("Payment recorded successfully")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
```

9. Create the exceptions in package **myexceptions** and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,
    - **CarNotFoundException**: throw this exception when user enters an invalid car id which doesn't exist in db.
    - **LeaseNotFoundException**: throw this exception when user enters an invalid lease id which doesn't exist in db.
    - **CustomerrNotFoundException**: throw this exception when user enters an invalid customer id which doesn't exist in db.

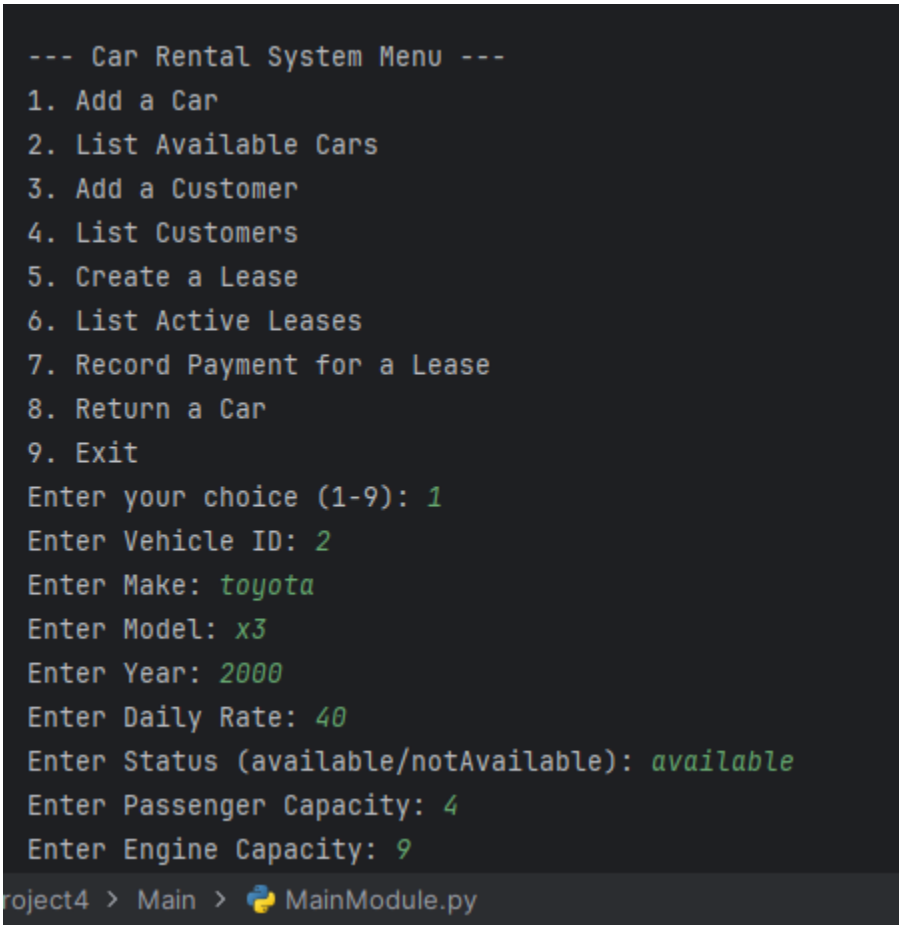
Unit Testing:

10. Create Unit test cases for **Ecommerce System** are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:
    - Write test case to test car created successfully or not.
    - Write test case to test lease is created successfully or not.
    - Write test case to test lease is retrieved successfully or not.
    - write test case to test exception is thrown correctly or not when customer id or car id or lease id not found in database.

```
from datetime import date
from dao.service.ICarLeaseRepositoryImpl import ICarLeaseRepositoryImpl
from entity.model.Customer import Customer
from entity.model.Vehicle import Vehicle

def print_menu():
    print("1. Add a Car")
    print("2. List Available Cars")
    print("3. Add a Customer")
    print("4. List Customers")
    print("5. Create a Lease")
```

```
    print("6. List Active Leases")
    print("7. Record Payment for a Lease")
    print("8. Return a Car")
    print("9. Exit")

def main():
    # Replace with your actual MySQL connection details
    connection_params = {
        "host": "localhost",
        "user": "root",
        "password": "root",
        "database": "CarRentalSystem"
    }

    car_repository = ICarLeaseRepositoryImpl(connection_params)

    while True:
        print("\n--- Car Rental System Menu ---")
        print_menu()
        choice = input("Enter your choice (1-9): ")

        if choice == "1":
            # Example: Adding a Car
            new_car = Vehicle(vehicleID=int(input("Enter Vehicle ID: ")),
                              make=input("Enter Make: "),
                              model=input("Enter Model: "),
                              year=int(input("Enter Year: ")),
                              dailyRate=float(input("Enter Daily Rate: ")),
                              status=input("Enter Status
(available/notAvailable): "),
                              passengerCapacity=int(input("Enter Passenger
Capacity: ")),
                              engineCapacity=int(input("Enter Engine
Capacity: ")))

            car_repository.addCar(new_car)
            print("Car added successfully")

        elif choice == "2":
            # Example: Listing Available Cars
            available_cars = car_repository.listAvailableCars()
            print("Available Cars:")
            for car in available_cars:
                print(car)

        elif choice == "3":
            # Example: Adding a Customer
            new_customer = Customer(customerID=int(input("Enter Customer ID:
")),
                                    firstName=input("Enter First Name: "),
                                    lastName=input("Enter Last Name: "),
                                    email=input("Enter Email: "),
                                    phoneNumber=input("Enter Phone Number:
"))

            car_repository.addCustomer(new_customer)
```

```
            print("Customer added successfully")

        elif choice == "4":
            # Example: Listing Customers
            customers = car_repository.listCustomers()
            print("Customers:")
            for customer in customers:
                print(customer)

        elif choice == "5":
            # Example: Creating a Lease
            start_date = date(2024, 2, 10)
            end_date = date(2024, 2, 20)
            new_lease =
car_repository.createLease(customerID=int(input("Enter Customer ID: ")),
                                                    carID=int(input("Enter Car
ID: ")),
                                                    startDate=start_date,
                                                    endDate=end_date)
            print("Lease created:", new_lease)

        elif choice == "6":
            # Example: Listing Active Leases
            active_leases = car_repository.listActiveLeases()
            print("Active Leases:")
            for lease in active_leases:
                print(lease)

        elif choice == "7":
            # Example: Recording Payment for a Lease
            lease id = int(input("Enter Lease ID: "))
            payment_amount = float(input("Enter Payment Amount: "))
            car_repository.recordPayment(leaseID=lease_id,
amount=payment_amount)
            print("Payment recorded successfully")

        elif choice == "8":
            # Example: Returning a Car
            lease_id = int(input("Enter Lease ID: "))
            returned_lease = car_repository.returnCar(leaseID=lease_id)
            print("Car returned. Updated Lease information:", returned_lease)

        elif choice == "9":
            # Exit the program
            print("Exiting Car Rental System. Goodbye!")
            car_repository.close_connection()
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 9.")

if __name__ == "__main__":
    main()
```


--- Car Rental System Menu ---
1. Add a Car
2. List Available Cars
3. Add a Customer
4. List Customers
5. Create a Lease
6. List Active Leases
7. Record Payment for a Lease
8. Return a Car
9. Exit
Enter your choice (1-9): 1
Enter Vehicle ID: 2
Enter Make: toyota
Enter Model: x3
Enter Year: 2000
Enter Daily Rate: 40
Enter Status (available/notAvailable): available
Enter Passenger Capacity: 4
Enter Engine Capacity: 9
roject4 > Main > MainModule.py