Case Study : Car Rental System
By Astha Raj

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

```
Database changed
mysql> CREATE TABLE Vehicle (
    ->     vehicleID INT PRIMARY KEY AUTO_INCREMENT,
    ->     make VARCHAR(255),
    ->     model VARCHAR(255),
    ->     year INT,
    ->     dailyRate DECIMAL(10, 2),
    ->     status VARCHAR(20),
    ->     passengerCapacity INT,
    ->     engineCapacity INT
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Customer (
    ->     customerID INT PRIMARY KEY AUTO_INCREMENT,
    ->     firstName VARCHAR(255),
    ->     lastName VARCHAR(255),
    ->     email VARCHAR(255),
    ->     phoneNumber VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE Lease (
    ->     leaseID INT PRIMARY KEY AUTO_INCREMENT,
    ->     vehicleID INT,
    ->     customerID INT,
    ->     startDate DATE,
    ->     endDate DATE,
    ->     type VARCHAR(20),
    ->     FOREIGN KEY (vehicleID) REFERENCES Vehicle(vehicleID),
    ->     FOREIGN KEY (customerID) REFERENCES Customer(customerID)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE TABLE Payment (
    ->     paymentID INT PRIMARY KEY AUTO_INCREMENT,
    ->     leaseID INT,
    ->     paymentDate DATE,
    ->     amount DECIMAL(10, 2),
    ->     FOREIGN KEY (leaseID) REFERENCES Lease(leaseID)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> _
```

MySQL 8.0 Command Line Client

```
Records: 10  Duplicates: 0  Warnings: 0

mysql> select * from customer;
+------------+-----------+----------+----------------------+--------------+
| customerID | firstName | lastName | email                | phoneNumber  |
+------------+-----------+----------+----------------------+--------------+
|          1 | John      | Doe      | johndoe@example.com  | 555-555-5555 |
|          2 | Jane      | Smith    | janesmith@example.com | 555-123-4567 |
|          3 | Robert    | Johnson  | robert@example.com   | 555-789-1234 |
|          4 | Sarah     | Brown    | sarah@example.com    | 555-456-7890 |
|          5 | David     | Lee      | david@example.com    | 555-987-6543 |
|          6 | Laura     | Hall     | laura@example.com    | 555-234-5678 |
|          7 | Michael   | Davis    | michael@example.com  | 555-876-5432 |
|          8 | Emma      | Wilson   | emma@example.com     | 555-432-1098 |
|          9 | William   | Taylor   | william@example.com  | 555-321-6547 |
|         10 | Olivia    | Adams    | olivia@example.com   | 555-765-4321 |
+------------+-----------+----------+----------------------+--------------+
10 rows in set (0.00 sec)

mysql> select * from vehicle;
+----------+----------+----------+------+----------+--------------+------------------+----------------+
| vehicleID | make     | model    | year | dailyRate | status       | passengerCapacity | engineCapacity |
+----------+----------+----------+------+----------+--------------+------------------+----------------+
|         1 | Toyota   | Camry    | 2022 |    50.00 | available    |                4 |           1450 |
|         2 | Honda    | Civic    | 2023 |    45.00 | available    |                7 |           1500 |
|         3 | Ford     | Focus    | 2022 |    48.00 | notAvailable |                4 |           1400 |
|         4 | Nissan   | Altima   | 2023 |    52.00 | available    |                7 |           1200 |
|         5 | Chevrolet | Malibu  | 2022 |    47.00 | available    |                4 |           1800 |
|         6 | Hyundai  | Sonata   | 2023 |    49.00 | notAvailable |                7 |           1400 |
|         7 | BMW      | 3 Series | 2023 |    60.00 | available    |                7 |           2499 |
|         8 | Mercedes | C-Class  | 2022 |    58.00 | available    |                8 |           2599 |
|         9 | Audi     | A4       | 2022 |    55.00 | notAvailable |                4 |           2500 |
|        10 | Lexus    | ES       | 2023 |    54.00 | available    |                4 |           2500 |
+----------+----------+----------+------+----------+--------------+------------------+----------------+
10 rows in set (0.00 sec)

mysql> select * from lease;
+---------+----------+----------+----------+----------+---------+
| leaseID | vehicleID | customerID | startDate | endDate | type    |
+---------+----------+----------+----------+----------+---------+
|       1 |        1 |        1 | 2023-01-01 | 2023-01-05 | Daily   |
|       2 |        2 |        2 | 2023-02-15 | 2023-02-28 | Monthly |
|       3 |        3 |        3 | 2023-03-10 | 2023-03-15 | Daily   |
|       4 |        4 |        4 | 2023-04-20 | 2023-04-30 | Monthly |
|       5 |        5 |        5 | 2023-05-05 | 2023-05-10 | Daily   |
|       6 |        4 |        3 | 2023-06-15 | 2023-06-30 | Monthly |
|       7 |        7 |        7 | 2023-07-01 | 2023-07-10 | Daily   |
|       8 |        8 |        8 | 2023-08-12 | 2023-08-15 | Monthly |
|       9 |        3 |        3 | 2023-09-07 | 2023-09-10 | Daily   |
|      10 |       10 |       10 | 2023-10-10 | 2023-10-31 | Monthly |
```

MySQL 8.0 Command Line Client

2	Honda	Civic	2023	45.00	available	7	1500
3	Ford	Focus	2022	48.00	notAvailable	4	1400
4	Nissan	Altima	2023	52.00	available	7	1200
5	Chevrolet	Malibu	2022	47.00	available	4	1800
6	Hyundai	Sonata	2023	49.00	notAvailable	7	1400
7	BMW	3 Series	2023	60.00	available	7	2499
8	Mercedes	C-Class	2022	58.00	available	8	2599
9	Audi	A4	2022	55.00	notAvailable	4	2500
10	Lexus	ES	2023	54.00	available	4	2500

10 rows in set (0.00 sec)

```
mysql> select * from lease;
```

leaseID	vehicleID	customerID	startDate	endDate	type
1	1	1	2023-01-01	2023-01-05	Daily
2	2	2	2023-02-15	2023-02-28	Monthly
3	3	3	2023-03-10	2023-03-15	Daily
4	4	4	2023-04-20	2023-04-30	Monthly
5	5	5	2023-05-05	2023-05-10	Daily
6	4	3	2023-06-15	2023-06-30	Monthly
7	7	7	2023-07-01	2023-07-10	Daily
8	8	8	2023-08-12	2023-08-15	Monthly
9	3	3	2023-09-07	2023-09-10	Daily
10	10	10	2023-10-10	2023-10-31	Monthly

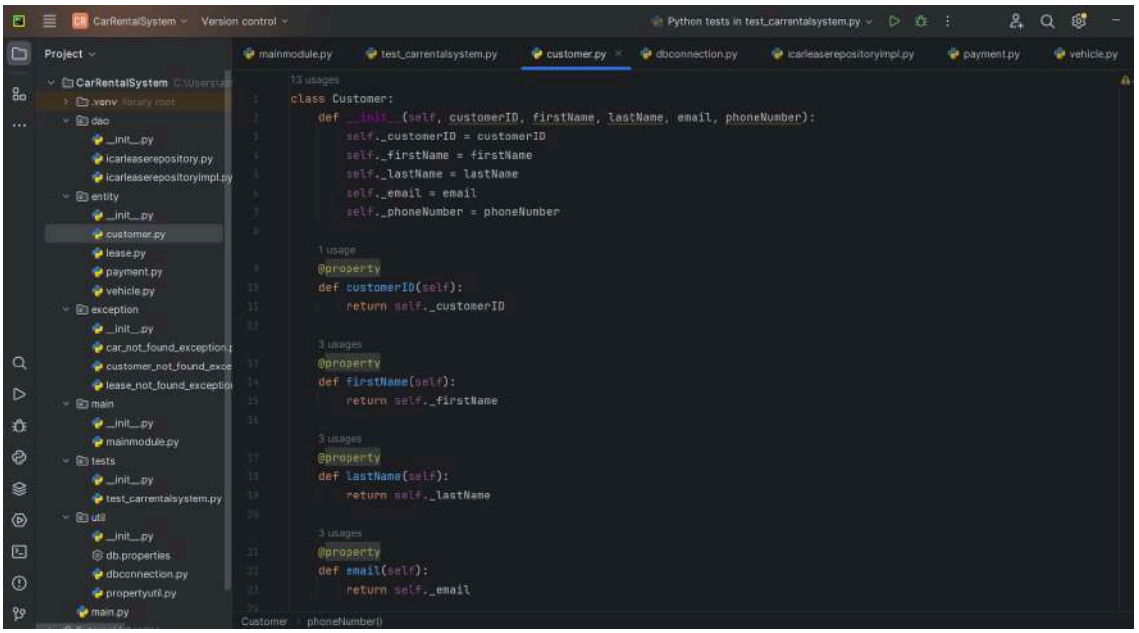
10 rows in set (0.00 sec)

```
mysql> select * from payment;
```

paymentID	leaseID	paymentDate	amount
1	1	2023-01-03	200.00
2	2	2023-02-20	1000.00
3	3	2023-03-12	75.00
4	4	2023-04-12	900.00
5	5	2023-05-12	60.00
6	6	2023-06-12	1200.00
7	7	2023-07-12	40.00
8	8	2023-08-14	1100.00
9	9	2023-09-09	80.00
10	10	2023-10-25	1500.00

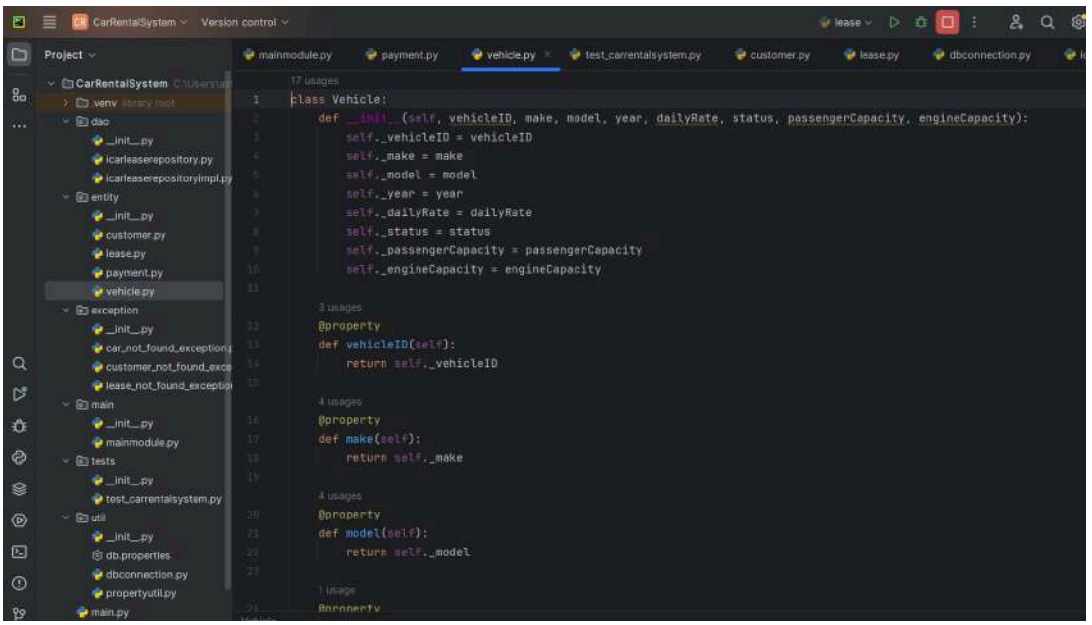
10 rows in set (0.00 sec)

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



```python
class Lease:
    def __init__(self, leaseID, vehicleID, customerID, startDate, endDate, type):
        self._leaseID = leaseID
        self._vehicleID = vehicleID
        self._customerID = customerID
        self._startDate = startDate
        self._endDate = endDate
        self._type = type


    @property
    def leaseID(self):
        return self._leaseID

    @property
    def vehicleID(self):
        return self._vehicleID

    @property
    def customerID(self):
        return self._customerID

    @property
    def startDate(self):
        return self._startDate
```

```python
class Payment:
    def __init__(self, paymentID, leaseID, paymentDate, amount):
        self._paymentID = paymentID
        self._leaseID = leaseID
        self._paymentDate = paymentDate
        self._amount = amount

    @property
    def paymentID(self):
        return self._paymentID

    @property
    def leaseID(self):
        return self._leaseID
    @property
    def paymentDate(self):
        return self._paymentDate

    @property
    def amount(self):
        return self._amount
```



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 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

```python
from abc import ABC, abstractmethod
from typing import List
from entity.vehicle import Vehicle
from entity.customer import Customer
from entity.lease import Lease

class ICarLeaseRepository(ABC):
    @abstractmethod
    def addCar(self, car: Vehicle):
        pass

    @abstractmethod
    def removeCar(self, carID: int):
        pass

    @abstractmethod
    def listAvailableCars(self) -> List[Vehicle]:
        pass

    @abstractmethod
    def listRentedCars(self) -> List[Vehicle]:
        pass

    @abstractmethod
    def findCarById(self, carID: int) -> Vehicle:
        pass

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

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

    @abstractmethod
    def listCustomers(self) -> List[Customer]:
        pass

    @abstractmethod
    def findCustomerById(self, customerID: int) -> Customer:
        pass

    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate, endDate, type) -> Lease:
        pass

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

    @abstractmethod
    def findLeaseById(self, leaseID: int) -> Lease:
        pass
```

```python
        pass

    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate, endDate, type) -> Lease:
        pass

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

    @abstractmethod
    def findLeaseById(self, leaseID: int) -> Lease:
        pass

    @abstractmethod
    def listActiveLeases(self) -> List[Lease]:
        pass

    @abstractmethod
    def listLeaseHistory(self) -> List[Lease]:
        pass

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

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

```python
from dao.icarleaserepository import ICarLeaseRepository
from entity.vehicle import Vehicle
from entity.customer import Customer
from entity.lease import Lease
from entity.payment import Payment
from exception.car_not_found_exception import CarNotFoundException
from exception.customer_not_found_exception import CustomerNotFoundException
from exception.lease_not_found_exception import LeaseNotFoundException
from util.dbconnection import DBConnection
from util.propertyutil import propertyUtil
from typing import List
from datetime import datetime


4 usages
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self, connection=None):
        self.connection = connection


    2 usages
    def addCar(self, car: Vehicle):
        cursor = self.connection.cursor()

        try:
            cursor.execute(
                "INSERT INTO vehicle (make, model, year, dailyRate, status, passengerCapacity, engineCapacity) "
                "VALUES (%s, %s, %s, %s, %s, %s, %s)",
                (car.make, car.model, car.year, car.dailyRate, car.status,
```

```python
        cursor = self.connection.cursor()

        try:
            cursor.execute("DELETE FROM vehicle WHERE vehicleID=%s", (carID,))
            self.connection.commit()
        finally:
            cursor.close()

    1 usage
    def listAvailableCars(self) -> List[Vehicle]:
        cursor = self.connection.cursor()

        try:
            cursor.execute("SELECT * FROM vehicle WHERE status='available'")
            rows = cursor.fetchall()

            cars = []
            for row in rows:
                car = Vehicle(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
                cars.append(car)

            return cars
        finally:
            cursor.close()

    1 usage
    def listRentedCars(self) -> List[Vehicle]:
```

```python
    1 usage
    def listRentedCars(self) -> List[Vehicle]:

        cursor = self.connection.cursor()

        try:
            cursor.execute("SELECT * FROM vehicle WHERE status='notAvailable'")
            rows = cursor.fetchall()

            cars = []
            for row in rows:
                car = Vehicle(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
                cars.append(car)

            return cars
        finally:
            cursor.close()

    3 usages
    def findCarById(self, carID: int) -> Vehicle:

        cursor = self.connection.cursor()

        try:
            cursor.execute("SELECT * FROM vehicle WHERE vehicleID=%s", (carID,))
            row = cursor.fetchone()
```

```python
            if row:
                return Vehicle(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
            else:
                raise CarNotFoundException(f"Car with ID {carID} not found.")
        finally:
            cursor.close()

    1 usage
    def addCustomer(self, customer: Customer):

        cursor = self.connection.cursor()

        try:
            cursor.execute("INSERT INTO customer (firstName, lastName, email, phoneNumber) VALUES (%s, %s, %s, %s)",
                           (customer.firstName, customer.lastName, customer.email, customer.phoneNumber))
            self.connection.commit()
        finally:
            cursor.close()

    1 usage
    def removeCustomer(self, customerID: int):
        if not self.connection:
            raise ValueError("Connection is not provided.")

        cursor = self.connection.cursor()

        try:
            cursor.execute("DELETE FROM customer WHERE customerID=%s", (customerID,))
```

```python
        cursor = self.connection.cursor()

        try:
            cursor.execute("DELETE FROM customer WHERE customerID=%s", (customerID,))
            self.connection.commit()
        finally:
            cursor.close()

    1 usage
    def listCustomers(self) -> List[Customer]:

        cursor = self.connection.cursor()

        try:
            cursor.execute("SELECT * FROM customer")
            rows = cursor.fetchall()

            customers = []
            for row in rows:
                customer = Customer(row[0], row[1], row[2], row[3], row[4])
                customers.append(customer)

            return customers
        finally:
            cursor.close()

    2 usages
    def findCustomerById(self, customerID: int) -> Customer:
```

```python
    2 usages
    def findCustomerById(self, customerID: int) -> Customer:

        cursor = self.connection.cursor()

        try:
            cursor.execute("SELECT * FROM customer WHERE customerID=%s", (customerID,))
            row = cursor.fetchone()

            if row:
                return Customer(row[0], row[1], row[2], row[3], row[4])
            else:
                raise CustomerNotFoundException(f"Customer with ID {customerID} not found.")
        finally:
            cursor.close()

    2 usages
    def createLease(self, customerID: int, carID: int, startDate, endDate, lease_type) -> Lease:

        cursor = self.connection.cursor()

        try:
            cursor.execute("INSERT INTO lease (vehicleID, customerID, startDate, endDate, type ) VALUES (%s, %s, %s, %s, %s)",
                           (carID, customerID, startDate, endDate, lease_type))
            self.connection.commit()
            cursor.execute("SELECT LAST_INSERT_ID()")
            leaseID = cursor.fetchone()[0]

            return Lease(leaseID, carID, customerID, startDate, endDate
```

```python
            return Lease(leaseID, carID, customerID, startDate, endDate,
                         lease_type)
        finally:
            cursor.close()

    1 usage
    def returnCar(self, leaseID: int) -> Lease:


        cursor = self.connection.cursor()

        try:
            cursor.execute("UPDATE lease SET endDate = %s WHERE leaseID = %s",
                           (datetime.now().date(), leaseID))
            self.connection.commit()

            return self.findLeaseById(leaseID)
        finally:
            cursor.close()

    5 usages
    def findLeaseById(self, leaseID: int) -> Lease:
        cursor = self.connection.cursor()

        try:
            cursor.execute("SELECT * FROM lease WHERE leaseID = %s", (leaseID,))
            result = cursor.fetchone()
```

```python
            cursor.close()

    5 usages
    def findLeaseById(self, leaseID: int) -> Lease:
        cursor = self.connection.cursor()
        try:
            cursor.execute("SELECT * FROM lease WHERE leaseID = %s", (leaseID,))
            result = cursor.fetchone()

            if result:
                lease_id, car_id, customer_id, start_date, end_date, type = result
                return Lease(lease_id,  car_id, customer_id, start_date, end_date, type)
            else:
                raise LeaseNotFoundException(f"Lease with ID {leaseID} not found.")
        finally:
            cursor.close()

    1 usage
    def listActiveLeases(self) -> List[Lease]:
        cursor = self.connection.cursor()
        try:
            cursor.exe  Instance attribute connection of dao.icarleaserepositoryimpl.ICarLeaseRepositoryImpl  DATE()")
            rows = cursor.fetchall()  connection: Any = connection
            leases = []
            for row in rows:
                lease = Lease(row[0], row[1], row[2], row[3], row[4], row[5])
                leases.append(lease)

            return leases
```

```python
    def listLeaseHistory(self) -> List[Lease]:
        if not self.connection:
            raise ValueError("Connection is not provided.")
        cursor = self.connection.cursor()
        try:
            cursor.execute("SELECT * FROM lease")
            rows = cursor.fetchall()

            leases = []
            for row in rows:
                lease = Lease(row[0], row[1], row[2], row[3], row[4], row[5])
                leases.append(lease)
            return leases
        finally:
            cursor.close()

    1 usage
    def recordPayment(self, lease: Lease, amount: float):
        cursor = self.connection.cursor()
        try:
            print("Lease id", lease.leaseID)
            cursor.execute("INSERT INTO payment (leaseID, paymentDate, amount) VALUES (%s, %s, %s)",
                           (lease.leaseID, datetime.now().strftime("%Y-%m-%d"), amount))
            self.connection.commit()
        finally:
            cursor.close()
```

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 f ile. • 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.

```python
import mysql.connector
from mysql.connector import Error
from util.propertyutil import propertyUtil

0 usages
class DBConnection:

    connection = None

    2 usages
    @staticmethod
    def getConnection():
        if DBConnection.connection is None:
            try:
                connection_string = propertyUtil.getPropertyString()
                DBConnection.connection = mysql.connector.connect(**connection_string)

                if DBConnection.connection.is_connected():
                    print("Database connected successfully")

            except Error as e:
                print(f'Error : {e}')

        return DBConnection.connection
```

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.

• main

```python
from dao.icarleaserepositoryimpl import ICarLeaseRepositoryImpl
from entity.vehicle import Vehicle
from entity.customer import Customer
from exception.car_not_found_exception import CarNotFoundException
from exception.customer_not_found_exception import CustomerNotFoundException
from exception.lease_not_found_exception import LeaseNotFoundException
from util.dbconnection import DBConnection


3 usages
class MainModule:

    def __init__(self):
        self.car_repository = ICarLeaseRepositoryImpl(DBConnection.getConnection())

    1 usage
    def display_menu(self):
        print("\nCar Rental System Menu:")
        print("1. Add a Car")
        print("2. Remove a Car")
        print("3. List Available Cars")
        print("4. List Rented Cars")
        print("5. Find Car by ID")
        print("6. Add a Customer")
        print("7. Remove a Customer")
        print("8. List Customers")
        print("9. Find Customer by ID")
        print("10. Create Lease")
        print("11. Return Car")
```

```python
        print("13. List Lease History")
        print("14. Record Payment")
        print("15. Exit")

    1 usage
    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice (1-15): ")

            if choice == "15":
                print("Exiting Car Rental System. Goodbye!")
                break
            elif choice == "1":
                self.add_car()
            elif choice == "2":
                self.remove_car()
            elif choice == "3":
                self.list_available_cars()
            elif choice == "4":
                self.list_rented_cars()
            elif choice == "5":
                self.find_car_by_id()
            elif choice == "6":
                self.add_customer()
            elif choice == "7":
                self.remove_customer()
            elif choice == "8":
                self.list_customers()
```

```python
            elif choice == "8":
                self.list_customers()
            elif choice == "9":
                self.find_customer_by_id()
            elif choice == "10":
                self.create_lease()
            elif choice == "11":
                self.return_car()
            elif choice == "12":
                self.list_active_leases()
            elif choice == "13":
                self.list_lease_history()
            elif choice == "14":
                self.record_payment()
            else:
                print("Invalid choice. Please enter a valid option.")

    1 usage
    def add_car(self):
        print("Adding a new Car:")
        make = input("Enter Make: ")
        model = input("Enter Model: ")
        year = input("Enter Year: ")
        daily_rate = input("Enter Daily Rate: ")
        status = input("Enter Status (available/notAvailable): ")
        passenger_capacity = input("Enter Passenger Capacity: ")
        engine_capacity = input("Enter Engine Capacity: ")

        new_car = Vehicle( vehicleID: None, make, model, year, daily_rate, status, passenger_capacity, engine_capacity)
```

```python
        status = input("Enter Status (available/notAvailable): ")
        passenger_capacity = input("Enter Passenger Capacity: ")
        engine_capacity = input("Enter Engine Capacity: ")

        new_car = Vehicle( vehicleID: None, make, model, year, daily_rate, status, passenger_capacity, engine_capacity)
        self.car_repository.addCar(new_car)
        print("Great! Car added successfully.")

    1 usage
    def remove_car(self):
        print("Remove a Car:")
        car_id = int(input("Enter Car ID to remove: "))

        try:
            self.car_repository.removeCar(car_id)
            print("Car removed successfully.")
        except CarNotFoundException as e:
            print(f"Error: {e}")

    1 usage
    def list_available_cars(self):
        print("List Available Cars:")
        available_cars = self.car_repository.listAvailableCars()

        if available_cars:
            print("Available Cars:")
            for car in available_cars:
                print(f"{car.vehicleID}. {car.make} {car.model} - Status: {car.status}")
        else:
```

```python
                print(f"{car.vehicleID}. {car.make} {car.model} - Status: {car.status}")
        else:
            print("No available cars.")

    1 usage
    def list_rented_cars(self):
        print("List Rented Cars:")
        rented_cars = self.car_repository.listRentedCars()

        if rented_cars:
            print("Rented Cars:")
            for car in rented_cars:
                print(f"{car.vehicleID}. {car.make} {car.model} - Status: {car.status}")
        else:
            print("No rented cars.")

    1 usage
    def find_car_by_id(self):
        print("Find Car by ID:")
        car_id = int(input("Enter Car ID to find: "))

        try:
            found_car = self.car_repository.findCarById(car_id)
            print(f"Car Found: {found_car.make} {found_car.model} - Status: {found_car.status}")
        except CarNotFoundException as e:
            print(f"Error: {e}")

    1 usage
    def add_customer(self):
```

```python
    1 usage
    def add_customer(self):
        print("Add a Customer:")
        first_name = input("Enter First Name: ")
        last_name = input("Enter Last Name: ")
        email = input("Enter Email: ")
        phone_number = input("Enter Phone Number: ")

        new_customer = Customer(customerID: None, first_name, last_name, email, phone_number)
        self.car_repository.addCustomer(new_customer)
        print("Customer added successfully.")

    1 usage
    def remove_customer(self):
        print("Remove a Customer:")
        customer_id = int(input("Enter Customer ID to remove: "))
        try:
            self.car_repository.removeCustomer(customer_id)
            print("Customer removed successfully.")
        except CustomerNotFoundException as e:
            print(f"Error: {e}")

    1 usage
    def list_customers(self):
        print("List Customers:")
        customers = self.car_repository.listCustomers()
        if customers:
            print("Customers:")
            for customer in customers:
```

```python
        else:
            print("No customers.")

    def find_customer_by_id(self):
        print("Find Customer by ID:")
        customer_id = int(input("Enter Customer ID to find: "))

        try:
            found_customer = self.car_repository.findCustomerById(customer_id)
            print(f"Customer Found: {found_customer.firstName} {found_customer.lastName} - Email: {found_customer.email}")
        except CustomerNotFoundException as e:
            print(f"Error: {e}")

    def create_lease(self):
        print("Creating Lease:")
        customer_id = int(input("Enter Customer ID: "))
        car_id = int(input("Enter Vehicle ID: "))
        start_date = input("Enter Start Date (YYYY-MM-DD): ")
        end_date = input("Enter End Date (YYYY-MM-DD): ")
        lease_type = input("Input type as Daily or Monthly: ")

        try:
            created_lease = self.car_repository.createLease(customer_id, car_id, start_date, end_date, lease_type)
            print(f"Lease created successfully. Lease ID: {created_lease.leaseID}")
        except (CarNotFoundException, CustomerNotFoundException) as e:
            raise e
```

```python
            raise e

    def return_car(self):
        print("Return Car:")
        lease_id = int(input("Enter Lease ID to return the car: "))
        try:
            returned_lease = self.car_repository.returnCar(lease_id)
            print(f"Car returned successfully. Lease ID: {returned_lease.leaseID}")
        except LeaseNotFoundException as e:
            print(f"Error: {e}")

    def list_active_leases(self):
        print("List Active Leases:")
        active_leases = self.car_repository.listActiveLeases()

        if active_leases:
            print("Active Leases:")
            for lease in active_leases:
                print(f"Lease ID: {lease.leaseID} - Start Date: {lease.startDate} - End Date: {lease.endDate}")
        else:
            print("No active leases.")

    def list_lease_history(self):
        print("List Lease History:")
        lease_history = self.car_repository.listLeaseHistory()

        if lease_history:
```

```python
            print("Lease History:")
            for lease in lease_history:
                print(f"Lease ID: {lease.leaseID} - Start Date: {lease.startDate} - End Date: {lease.endDate}")
        else:
            print("No lease history.")

    1 usage
    def record_payment(self):
        print("Record Payment:")
        lease_id = int(input("Enter Lease ID for payment: "))
        amount = float(input("Enter Payment Amount: "))

        try:
            lease = self.car_repository.findLeaseById(lease_id)
            self.car_repository.recordPayment(lease, amount)
            print("Payment recorded successfully.")
        except LeaseNotFoundException as e:
            print(f"Error: {e}")


if __name__ == "__main__":
    main_module = MainModule()
    main_module.run()
```

OUTPUTS :

```
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 1
Adding a new Car:
Enter Make: Audi
Enter Model: A5
Enter Year: 2021
Enter Daily Rate: 55.00
Enter Status (available/notAvailable): available
Enter Passenger Capacity: 6
Enter Engine Capacity: 2000
Great! Car added successfully.

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
```

```
1. Toyota Camry - Status: available
2. Honda Civic - Status: available
4. Nissan Altima - Status: available
5. Chevrolet Malibu - Status: available
7. BMW 3 Series - Status: available
8. Mercedes C-Class - Status: available
10. Lexus ES - Status: available
11. Toyota Camry - Status: available
12. Audi A5 - Status: available

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15):
```

```
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 4
List Rented Cars:
Rented Cars:
3. Ford Focus - Status: notAvailable
6. Hyundai Sonata - Status: notAvailable
9. Audi A4 - Status: notAvailable

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
```

```
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 5
Find Car by ID:
Enter Car ID to find: 5
Car Found: Chevrolet Malibu - Status: available

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15):
```

```
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 6
Add a Customer:
Enter First Name: Astha
Enter Last Name: Raj
Enter Email: astha@gmail.com
Enter Phone Number: 212-252-4568
Customer added successfully.

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
```

```
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 8
List Customers:
Customers:
1. John Doe - Email: johndoe@example.com
2. Jane Smith - Email: janesmith@example.com
3. Robert Johnson - Email: robert@example.com
4. Sarah Brown - Email: sarah@example.com
5. David Lee - Email: david@example.com
6. Laura Hall - Email: laura@example.com
7. Michael Davis - Email: michael@example.com
8. Emma Wilson - Email: emma@example.com
9. William Taylor - Email: william@example.com
10. Olivia Adams - Email: olivia@example.com
11. Astha Raj - Email: astha@gmail.com

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
```

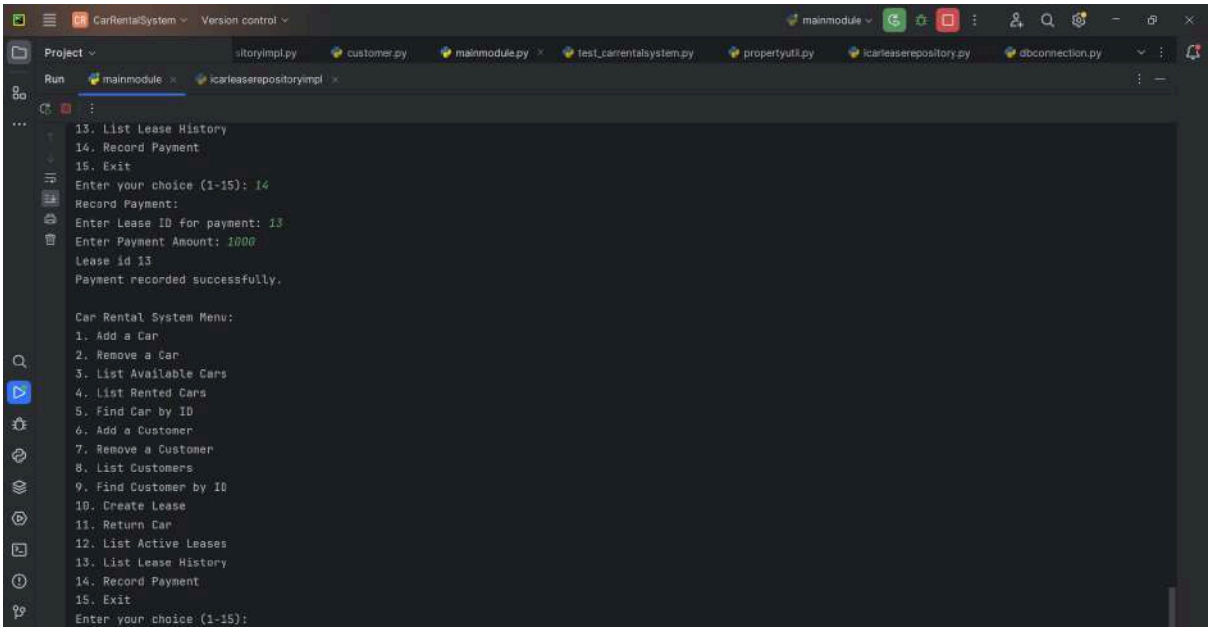
```
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 7
Remove a Customer:
Enter Customer ID to remove: 11
Customer removed successfully.

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15):
```

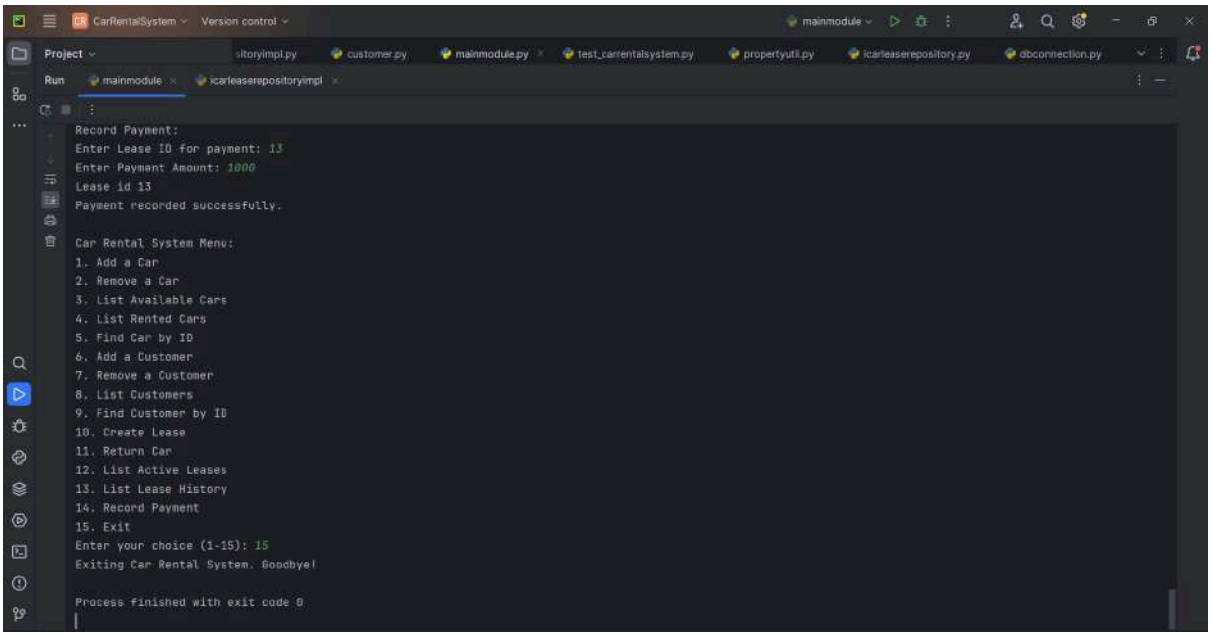
```
14. Record Payment
15. Exit
Enter your choice (1-15): 8
List Customers:
Customers:
1. John Doe - Email: johndoe@example.com
2. Jane Smith - Email: janesmith@example.com
3. Robert Johnson - Email: robert@example.com
4. Sarah Brown - Email: sarah@example.com
5. David Lee - Email: david@example.com
6. Laura Hall - Email: laura@example.com
7. Michael Davis - Email: michael@example.com
8. Emma Wilson - Email: emma@example.com
9. William Taylor - Email: william@example.com
10. Olivia Adams - Email: olivia@example.com

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
```

```
8. Emma Wilson - Email: emma@example.com
9. William Taylor - Email: william@example.com
10. Olivia Adams - Email: olivia@example.com

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 9
Find Customer by ID:
Enter Customer ID to find: 10
Customer Found: Olivia Adams - Email: olivia@example.com

Car Rental System Menu:
1. Add a Car
```

```
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 12
List Active Leases:
Active Leases:
Lease ID: 12 - Start Date: 2024-02-07 - End Date: 2024-12-12

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Add a Customer
7. Remove a Customer
8. List Customers
9. Find Customer by ID
10. Create Lease
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15):
```

```
11. Return Car
12. List Active Leases
13. List Lease History
14. Record Payment
15. Exit
Enter your choice (1-15): 13
List Lease History:
Lease History:
Lease ID: 1 - Start Date: 2023-01-01 - End Date: 2023-01-05
Lease ID: 2 - Start Date: 2023-02-15 - End Date: 2023-02-28
Lease ID: 3 - Start Date: 2023-03-10 - End Date: 2023-03-15
Lease ID: 4 - Start Date: 2023-04-20 - End Date: 2023-04-30
Lease ID: 5 - Start Date: 2023-05-05 - End Date: 2023-05-10
Lease ID: 6 - Start Date: 2023-06-15 - End Date: 2023-06-30
Lease ID: 7 - Start Date: 2023-07-01 - End Date: 2023-07-10
Lease ID: 8 - Start Date: 2023-08-12 - End Date: 2023-08-15
Lease ID: 9 - Start Date: 2023-09-07 - End Date: 2023-09-10
Lease ID: 10 - Start Date: 2023-10-10 - End Date: 2023-10-31
Lease ID: 11 - Start Date: 2024-02-07 - End Date: 2023-01-01
Lease ID: 12 - Start Date: 2024-02-07 - End Date: 2024-12-12
Lease ID: 13 - Start Date: 2024-02-08 - End Date: 2024-02-07

Car Rental System Menu:
1. Add a Car
2. Remove a Car
3. List Available Cars
4. List Rented Cars
```





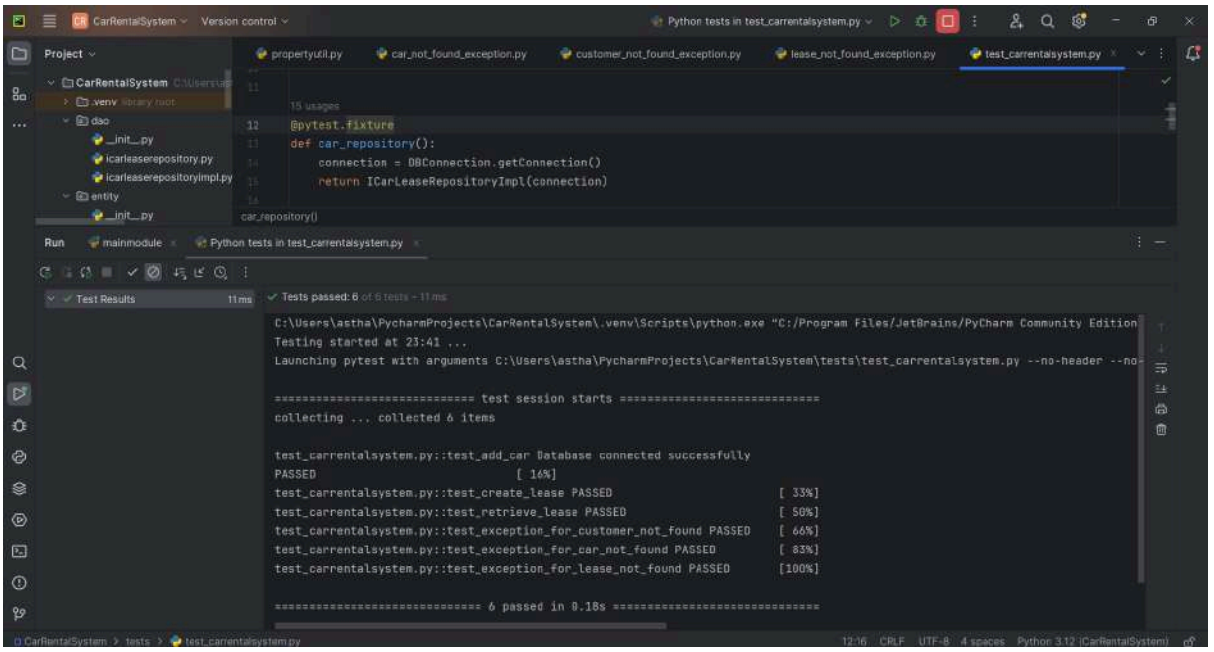
• Create a class MainModule and demonstrate the functionalities in a menu driven application.

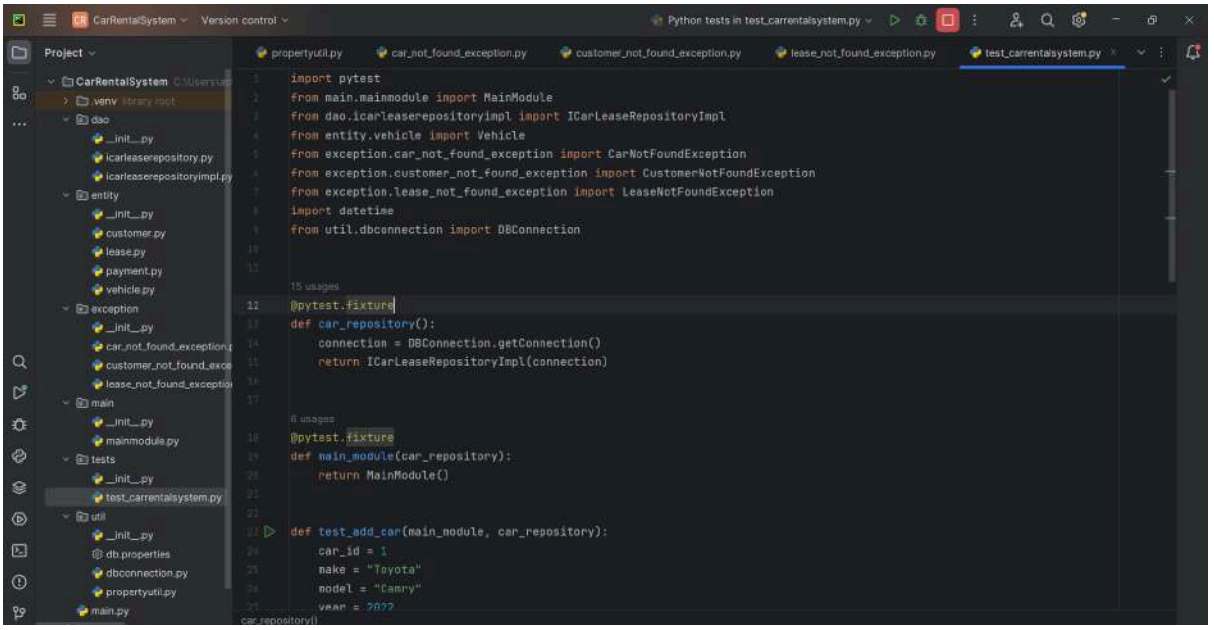
Unit Testing: 10.

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.





```python
def test_add_car(main_module, car_repository):
    car_id = 1
    make = "Toyota"
    model = "Camry"
    year = 2022
    daily_rate = 30
    status = "available"
    passenger_capacity = 5
    engine_capacity = 3

    new_car = Vehicle(car_id, make, model, year, daily_rate, status, passenger_capacity, engine_capacity)
    car_repository.addCar(new_car)

    assert car_repository.findCarById(new_car.vehicleID)


def test_create_lease(main_module, car_repository):
    customer_id = 1
    car_id = 1
    start_date = datetime.datetime.now().strftime("%Y-%m-%d")
    end_date = "2024-02-07"
    lease_type = "Monthly"

    created_lease = car_repository.createLease(customer_id, car_id, start_date, end_date, lease_type)

    assert car_repository.findLeaseById(created_lease.leaseID)


def test_retrieve_lease(main_module, car_repository):
```

```python
    lease_type = "Monthly"

    created_lease = car_repository.createLease(customer_id, car_id, start_date, end_date, lease_type)

    assert car_repository.findLeaseById(created_lease.leaseID)


def test_retrieve_lease(main_module, car_repository):
    lease_id = 1
    retrieved_lease = car_repository.findLeaseById(lease_id)
    assert retrieved_lease is not None


def test_exception_for_customer_not_found(main_module, car_repository):
    with pytest.raises(CustomerNotFoundException):
        car_repository.findCustomerById(9999)


def test_exception_for_car_not_found(main_module, car_repository):
    with pytest.raises(CarNotFoundException):
        car_repository.findCarById(9999)


def test_exception_for_lease_not_found(main_module, car_repository):
    with pytest.raises(LeaseNotFoundException):
        car_repository.findLeaseById(9999)
```

```
(.venv) PS C:\Users\astha\PycharmProjects\CarRentalSystem> pytest -k test_create_lease
========================================================================= test session starts =========================================================================
platform win32 -- Python 3.12.1, pytest-8.0.0, pluggy-1.4.0
rootdir: C:\Users\astha\PycharmProjects\CarRentalSystem
collected 6 items / 5 deselected / 1 selected

tests\test_carrentalsystem.py .                                                                                                                                      [100%]

============================================================================= 1 passed, 5 deselected in 0.14s =============================================================
(.venv) PS C:\Users\astha\PycharmProjects\CarRentalSystem> pytest -k test_add_car
========================================================================= test session starts =========================================================================
platform win32 -- Python 3.12.1, pytest-8.0.0, pluggy-1.4.0
rootdir: C:\Users\astha\PycharmProjects\CarRentalSystem
collected 6 items / 5 deselected / 1 selected

tests\test_carrentalsystem.py .                                                                                                                                      [100%]

============================================================================= 1 passed, 5 deselected in 0.13s =============================================================
(.venv) PS C:\Users\astha\PycharmProjects\CarRentalSystem> pytest -k test_retrieve_lease
========================================================================= test session starts =========================================================================
platform win32 -- Python 3.12.1, pytest-8.0.0, pluggy-1.4.0
rootdir: C:\Users\astha\PycharmProjects\CarRentalSystem
collected 6 items / 5 deselected / 1 selected

tests\test_carrentalsystem.py .                                                                                                                                      [100%]

============================================================================= 1 passed, 5 deselected in 0.42s =============================================================
(.venv) PS C:\Users\astha\PycharmProjects\CarRentalSystem>
```

THANK YOU