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

```
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
);
```

```
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 (0.06 sec)

mysql>
```

2. **Customer Table:**
- customerID (Primary Key)
- firstName
- lastName
- email
- phoneNumber

```
CREATE TABLE Customer (
```

```
    customerID INT PRIMARY KEY,
    firstName VARCHAR(255),
    lastName VARCHAR(255),
    email VARCHAR(255),
    phoneNumber VARCHAR(20)
);
```

```
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 (0.01 sec)
```

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)

```
CREATE TABLE Lease (
    leaseID INT PRIMARY KEY,
    vehicleID INT,
    customerID INT,
    startDate DATE,
    endDate DATE,
    type ENUM('Daily', 'Monthly'),
    FOREIGN KEY (vehicleID) REFERENCES Vehicle(vehicleID),
    FOREIGN KEY (customerID) REFERENCES Customer(customerID)
);
```

```
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 (0.01 sec)
```

4. **Payment Table:**
- paymentID (Primary Key)
- leaseID (Foreign Key referencing Lease Table)
- paymentDate
- amount

```
CREATE TABLE Payment (
    paymentID INT PRIMARY KEY,
    leaseID INT,
    paymentDate DATE,
    amount DECIMAL(10, 2),
    FOREIGN KEY (leaseID) REFERENCES Lease(leaseID)
);
```

```
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 | 2023-04-25  |  900.00 |
|         5 |       5 | 2023-05-07  |   60.00 |
|         6 |       6 | 2023-06-18  | 1200.00 |
|         7 |       7 | 2023-07-03  |   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 (0.01 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)**

entity

→customer.py

→lease.py

→vehicle.py

→payment.py

Customer class

```
class Customer:
    def __init__(self, customer_id, first_name, last_name, email,
phone_number):
        self.__customer_id = customer_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email
        self.__phone_number = phone_number

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_first_name(self):
        return self.__first_name

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def get_last_name(self):
        return self.__last_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def get_email(self):
        return self.__email

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

    def get_phone_number(self):
        return self.__phone_number

    def set_phone_number(self, phone_number):
        self.__phone_number = phone_number
```

lease class

```
class Lease:
    def __init__(self, lease_id, vehicle_id, customer_id, start_date,
end_date, lease_type):
        self.__lease_id = lease_id
        self.__vehicle_id = vehicle_id
        self.__customer_id = customer_id
        self.__start_date = start_date
```

```
        self.__end_date = end_date
        self.__lease_type = lease_type

    def get_lease_id(self):
        return self.__lease_id

    def set_lease_id(self, lease_id):
        self.__lease_id = lease_id

    def get_vehicle_id(self):
        return self.__vehicle_id

    def set_vehicle_id(self, vehicle_id):
        self.__vehicle_id = vehicle_id

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_start_date(self):
        return self.__start_date

    def set_start_date(self, start_date):
        self.__start_date = start_date

    def get_end_date(self):
        return self.__end_date

    def set_end_date(self, end_date):
        self.__end_date = end_date

    def get_lease_type(self):
        return self.__lease_type

    def set_lease_type(self, lease_type):
        self.__lease_type = lease_type
```

payment class

```
class Payment:
    def __init__(self, payment_id, lease_id, payment_date, amount):
        self.__payment_id = payment_id
        self.__lease_id = lease_id
        self.__payment_date = payment_date
        self.__amount = amount

    def get_payment_id(self):
        return self.__payment_id

    def set_payment_id(self, payment_id):
        self.__payment_id = payment_id

    def get_lease_id(self):
        return self.__lease_id

    def set_lease_id(self, lease_id):
        self.__lease_id = lease_id
```

```
    def get_payment_date(self):
        return self.__payment_date

    def set_payment_date(self, payment_date):
        self.__payment_date = payment_date

    def get_amount(self):
        return self.__amount

    def set_amount(self, amount):
        self.__amount = amount
```

vehicle class

```
class Vehicle:
    vehicleId=2
    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

    #vehicleId
    def set_vehicleId(self,vehicleId):
        self.__vehicleId=vehicleId
    def get_vehicleId(self):
        return self.vehicleId

    #make
    def set_make(self,make):
        self.__make=make
    def get_vehicleId(self):
        return self.__make

    #entity
    def set_model(self,model):
        self.__model=model
    def get_model(self):
        return self.__model

    #year
    def set_year(self,year):
        self.__year=year
    def get_year(self):
        return self.__year

    #dailyRate
    def set_dailyRate(self,dailyRate):
        self.__dailyRate=dailyRate
    def get_dailyRate(self):
        return self.__dailyRate

    #status
    def set_status(self,status):
```

```
        self.__status=status
    def get_status(self):
        return self.__status

    #passengerCapacity
    def set_passengerCapacity(self,passengerCapacity):
        self.__passengerCapacity=passengerCapacity
    def get_passengerCapacity(self):
        return self.__passengerCapacity

    #engineCapacity
    def set_engineCapacity(self,engineCapacity):
        self.__engineCapacity=engineCapacity
    def get_engineCapacity(self):
        return self.__engineCapacity
```

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.

```
from abc import ABC,abstractmethod

class CarManagement(ABC):
    @abstractmethod
    def addCar(self,carObj):
        pass
    @abstractmethod
    def removeCar(self,carId):
        pass
    @abstractmethod
    def listAvailableCars(self):
        pass
    @abstractmethod
    def listRentedCars(self):
        pass
    @abstractmethod
    def findCarById(self,carId):
        pass

class CustomerManagement(ABC):
    @abstractmethod
    def addCustomer(self, customerObj):
        pass
    @abstractmethod
    def removeCustomer(self, customerID):
        pass
    @abstractmethod
    def listCustomers(self):
        pass
    @abstractmethod
    def findCustomerById(self, customerID):
        pass

class LeaseManagement(ABC):
    @abstractmethod
    def createLease(self, customerID, carID, startDate, endDate):
        pass
    @abstractmethod
    def returnCar(self, leaseID):
        pass
```

```
    @abstractmethod
    def listActiveLeases(self):
        pass
    @abstractmethod
    def listLeaseHistory(self):
        pass

class PaymentHandling(ABC):
    @abstractmethod
    def recordPayment(self, lease, amount):
        pass
```

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

Car Management

```
class CarManagement(CarManagement):
    def addCar(self,carObj):
        cursor=ICarLeaseRepositoryImpl.connection.cursor()

        query=("insert into vehicle(make, model, year, dailyRate, status,
PassengerCapacity, engineCapacity)"
               "values(%s,%s,%s,%s,%s,%s,%s)")
        values=(
            carObj.get_make(),
            carObj.get_model(),
            carObj.get_year(),
            carObj.get_dailyRate(),
            carObj.get_status(),
            carObj.get_passengerCapacity(),
            carObj.get_engineCapacity()
        )

        cursor.execute(query,values)
        cursor.execute("commit")
```

```
Car Management Menu:
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Back to Main Menu
Enter your choice: 1
make:suzuki
model:model3
year:2024
dailyRate:15.0
status:notAvailable
passengerCapacity:5
engineCapacity:1250
Car Added successfully!
```

```
    def removeCar(self,carId):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query="delete from vehicle where vehicleId=%s"

        cursor.execute(query,(carId,))
        cursor.execute("commit")
```

```
Car Management Menu:
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Back to Main Menu
Enter your choice: 2
CarId:12
Car removed successfully!
```

```
    def listAvailableCars(self):
        availCar=[]
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query='select * from vehicle where status="available"'

        cursor.execute(query)
        rows=cursor.fetchall()

        print("Available Cars:")
        for row in rows:
            availCar.append((row[0],row[1]))

        print(availCar)
```

```
Car Management Menu:
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Back to Main Menu
Enter your choice: 3
Available Cars:
[(1, 'Toyota'), (2, 'Honda'), (4, 'Nissan'), (5, 'Chevrolet'), (7, 'BMW'), (8, 'Mercedes'), (10, 'Lexus')]
```

```
    def listRentedCars(self):
        rentCar = []
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query = ("SELECT v.* "
                 "FROM Vehicle v "
                 "JOIN Lease l ON v.vehicleID = l.vehicleID "
                 "WHERE l.endDate >= CURDATE();"
                 )
```

```
        cursor.execute(query)
        rows = cursor.fetchall()

        print("Rented Cars:")
        for row in rows:
            rentCar.append((row[0], row[1]))

        print(rentCar)
```

```
Car Management Menu:
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Back to Main Menu
Enter your choice: 4
Rented Cars:
[(5, 'Chevrolet'), (4, 'Nissan'), (7, 'BMW'), (8, 'Mercedes'), (3, 'Ford'), (10, 'Lexus'), (3, 'Ford')]
```

```
    def findCarById(self,carId):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query="select * from vehicle where vehicleId=%s"

        cursor.execute(query,(carId,))
        rows=cursor.fetchall()

        if rows is not None and len(rows) > 0:
            for row in rows:
                print("vehicleID:", row[0],
                      "make:", row[1],
                      "model:", row[2],
                      "year:", row[3],
                      "dailyRate:", row[4],
                      "status:", row[5],
                      "passengerCapacity:", row[6],
                      "engineCapacity:", row[7]
                      )
        else:
            raise CarNotFoundException()
```

```
Car Management Menu:
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
6. Back to Main Menu
Enter your choice: 5
CarId:14
CarId is not found!
```

Customer Management

```
class CustomerManagement(CustomerManagement):
    def addCustomer(self,custObj):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query=(
            "insert into customer (firstName, lastName, email,
phoneNumber)"
            "values(%s,%s,%s,%s)"
        )
        values=(
            custObj.get_first_name(),
            custObj.get_last_name(),
            custObj.get_email(),
            custObj.get_phone_number()
        )

        cursor.execute(query,values)
        cursor.execute("commit")
```

```
Customer Management Menu:
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Back to Main Menu
Enter your choice: 1
FirstName:ambika
SecondName:krishna
Email:ambika27@gmail.com
PhoneNumeber:9842466787
Customer Added successfully!
```

```
    def removeCustomer(self, customerID):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query="delete from customer where customerId=%s"

        cursor.execute(query,(customerID,))
        cursor.execute("commit")
```

```
Customer Management Menu:
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Back to Main Menu
Enter your choice: 2
CustomerId:11
Customer Removed successfully!
```

```
    def listCustomers(self):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query = 'select * from customer'

        cursor.execute(query)
        rows = cursor.fetchall()

        print("Customer Details:")
        for row in rows:
            print(row)
```

```
Customer Management Menu:
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Back to Main Menu
Enter your choice: 3
Customer Details:
(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')
(12, 'vaishu', 'devaraj', 'vaishu282gmaul.com', '9824355678')
(13, 'ambika', 'krishna', 'ambika27@gmail.com', '9842466787')
```

```
    def findCustomerById(self, customerID):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query="select * from customer where customerId=%s"

        cursor.execute(query,(customerID,))
        rows=cursor.fetchall()

        if rows is not None and len(rows) > 0:
            for row in rows:
                print(row)
        else:
            raise CustomerNotFoundException()
```

```
Customer Management Menu:
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Back to Main Menu
Enter your choice: 4
CustomerId:12
(12, 'vaishu', 'devaraj', 'vaishu282gmaul.com', '9824355678')
```

Lease Management

```
class LeaseManagement(LeaseManagement):
    def createLease(self,carID,customerID, startDate, endDate,type):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query="insert into lease (vehicleID, customerID, startDate,
endDate, type)values(%s,%s,%s,%s,%s)"

        values=(carID,customerID,startDate,endDate,type)

        cursor.execute(query,values)
        cursor.execute("commit")

    def returnCar(self, leaseID):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query = "select * from lease where leaseId=%s"

        cursor.execute(query, (leaseID,))
        rows=cursor.fetchall()

        if rows is not None and len(rows) > 0:
            for row in rows:
                print(row)
        else:
            raise LeaseNotFoundException()
```

```
Lease Management Menu:
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
5. Back to Main Menu
Enter your choice: 1
CarId:3
CustomerId:5
StartDate:2024-07-20
endDate:2024-08-19
Daily/Monthly:Monthly
Lease created successfully!
```

```
    def listActiveLeases(self):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        cursor.execute("SELECT * FROM Lease WHERE endDate >= CURDATE()")
        rows = cursor.fetchall()

        for row in rows:
            print(row)
```

```
Lease Management Menu:
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
5. Back to Main Menu
Enter your choice: 2
LeaseId:3
(3, 3, 3, datetime.date(2024, 3, 10), datetime.date(2024, 3, 15), 'Daily')
```

```
    def listLeaseHistory(self):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        cursor.execute("SELECT * FROM Lease WHERE endDate < CURDATE()")
        rows = cursor.fetchall()

        for row in rows:
            print(row)
```

```
Lease Management Menu:
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
5. Back to Main Menu
Enter your choice: 4
(1, 1, 1, datetime.date(2024, 1, 1), datetime.date(2024, 1, 5), 'Daily')
(2, 2, 2, datetime.date(2024, 2, 15), datetime.date(2024, 2, 28), 'Monthly')
(3, 3, 3, datetime.date(2024, 3, 10), datetime.date(2024, 3, 15), 'Daily')
(4, 4, 4, datetime.date(2024, 4, 20), datetime.date(2024, 4, 30), 'Monthly')
(12, 2, 4, datetime.date(2024, 2, 24), datetime.date(2024, 2, 26), 'Daily')
```

Payment Handling

```
class PaymentHandling(PaymentHandling):
    def recordPayment(self, lease, amount):
        cursor = ICarLeaseRepositoryImpl.connection.cursor()

        query="insert into payment (leaseID, paymentDate,
amount)values(%s,curdate(),%s)"
        values=(lease,amount)

        cursor.execute(query,values)
        cursor.execute("commit")
```

```
Payment Handling Menu:
1. Record Payment
2. Back to Main Menu
Enter your choice: 1
LeaseId:11
Amount:2000
Payment successfully!
```

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.

```
import mysql.connector

class DBConnection:
    connection=None
    @staticmethod
    def getConnection():
        f=open("propertyFile","r")
        lines=f.readlines()
        host=lines[0].strip()
        username=lines[1].strip()
        password=lines[2].strip()
        database=lines[3].strip()

        DBConnection.connection=mysql.connector.connect(
            host=host,
```

```
            user=username,
            password=password,
            database=database
        )

        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.

```
class CarNotFoundException(Exception):
    def __init__(self):
        super().__init__("CarId is not found!")

class LeaseNotFoundException(Exception):
    def __init__(self):
        super().__init__("LeaseId is not found!")

class CustomerNotFoundException(Exception):
    def __init__(self):
        super().__init__("CustomerId is not found!")
```

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.

```
import unittest,datetime
from dao.ICarLeaseRepositoryImpl import ICarLeaseRepositoryImpl
from entity.vehicle import Vehicle
from myExceptions.exceptions import CarNotFoundException,
CustomerNotFoundException, LeaseNotFoundException


class TestEcommerceSystem(unittest.TestCase):

    def setUp(self):
        self.repo = ICarLeaseRepositoryImpl()


    def test_car_creation(self):
        cursor = self.repo.connection.cursor()
        cursor.execute("SELECT MAX(vehicleID) FROM vehicle")
        max_id = cursor.fetchone()[0]
        carID = max_id + 1 if max_id is not None else 1

        carObj=Vehicle(carID,'Toyota','Camry',2022,50,'available',5,2.5)
        innerObj=self.repo.CarManagement()
        innerObj.addCar(carObj)
```

```
        self.assertIn((carID, "Toyota"), innerObj.listAvailableCars())


    def test_lease_creation(self):

        carID = 1
        customerID = 1
        startDate = "2024-05-01"
        endDate = "2024-05-15"
        lease_type = "Daily"

        innerObj=self.repo.LeaseManagement()
        innerObj.createLease(carID, customerID, startDate, endDate,
lease_type)

        self.assertIn((1, 1, datetime.date(2024,5,1),
datetime.date(2024,5,15), lease_type), innerObj.listActiveLeases())


    def test_lease_retrieval(self):
        leaseID = 1
        innerObj=self.repo.LeaseManagement()
        lease_details = innerObj.returnCar(leaseID)

        self.assertEqual(lease_details, (1, 1, 1, datetime.date(2024, 1,
1), datetime.date(2024, 1, 5), 'Daily'))


    def test_exception_handling(self):
        # Test CarNotFoundException
        with self.assertRaises(CarNotFoundException):
            innerObj=self.repo.CarManagement()
            innerObj.findCarById(100)

        # Test CustomerNotFoundException
        with self.assertRaises(CustomerNotFoundException):
            innerObj=self.repo.CustomerManagement()
            innerObj.findCustomerById(100)

        # Test LeaseNotFoundException
        with self.assertRaises(LeaseNotFoundException):
            innerObj=self.repo.LeaseManagement()
            innerObj.returnCar(100)

if __name__ == '__main__':
    unittest.main()
```

```
C:\Users\ambik>cd C:\Users\ambik\PycharmProjects\CarRental\test

C:\Users\ambik\PycharmProjects\CarRental\test>python -m pytest
============================================= test session starts =============================================
platform win32 -- Python 3.12.3, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\ambik\PycharmProjects\CarRental\test
collected 4 items

test_carRental.py ....                                                                              [100%]

============================================== 4 passed in 0.38s ==============================================

C:\Users\ambik\PycharmProjects\CarRental\test>
```