

Hexaware Technologies

Case Study : Car Rental System

Name - Astha Raj

The main focus of this case study is to implement core functionalities like Car Management, Customer Management, Lease Management and Payment Handling.

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

Schema Design:

-> Created a database named carrentalsystem.



MySQL 8.0 Command Line Client

```
mysql> create database carrentalsystem;  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> use carrentalsystem;  
Database changed
```

```
1. CREATE TABLE vehicle (
```

1. Vehicle Table:

- vehicleID (Primary Key)
- make
- model
- year
- dailyRate
- status (Available, NotAvailable)
- passengerCapacity
- engineCapacity

```
mysql> CREATE TABLE Vehicle (  
->   vehicleID INT PRIMARY KEY AUTO_INCREMENT,  
->   make VARCHAR(100),  
->   model VARCHAR(100),  
->   year INT,  
->   dailyRate DECIMAL(10, 2),  
->   status ENUM('Available','NotAvailable'),  
->   passengerCapacity INT,  
->   engineCapacity INT  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

-> Inserted 10 Records also:

```
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)
```

2. Customer Table:

- customerID (Primary Key)
- firstName
- lastName
- email
- phoneNumber

```
mysql> CREATE TABLE Customer (
->     customerID INT PRIMARY KEY AUTO_INCREMENT,
->     firstName VARCHAR(100),
->     lastName VARCHAR(100),
->     email VARCHAR(100),
->     phoneNumber VARCHAR(20)
-> );
Query OK, 0 rows affected (0.01 sec)
```

-> Inserted 10 Records also:

```
MySQL 8.0 Command Line Client
mysql> select * from customer;
```

customerID	firstName	lastName	email	phoneNumber
1	John	Doe	john.doe@example.com	555-555-5555
2	Jane	Smith	jane.smith@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)
```

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)

```
mysql> CREATE TABLE Lease (
->     leaseID INT PRIMARY KEY AUTO_INCREMENT,
->     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 (0.03 sec)
```

-> Inserted 10 Records

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

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

```
10 rows in set (0.00 sec)
```

4. Payment Table:

- paymentID (Primary Key)
- leaseID (Foreign Key referencing Lease Table)
- paymentDate
- amount

```
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 (0.02 sec)

mysql> _
```

-> Inserted 10 records also

```
MySQL 8.0 Command Line Client

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)
```

The following **Directory structure** is to be followed in the application.

- **entity/model**

- Create entity classes in this package. All entity class should not have any business logic.

- **dao**

- Create Service Provider interface to showcase functionalities.
- Create the implementation class for the above interface with db interaction.

- **exception**

- Create user defined exceptions in this package and handle exceptions whenever needed.

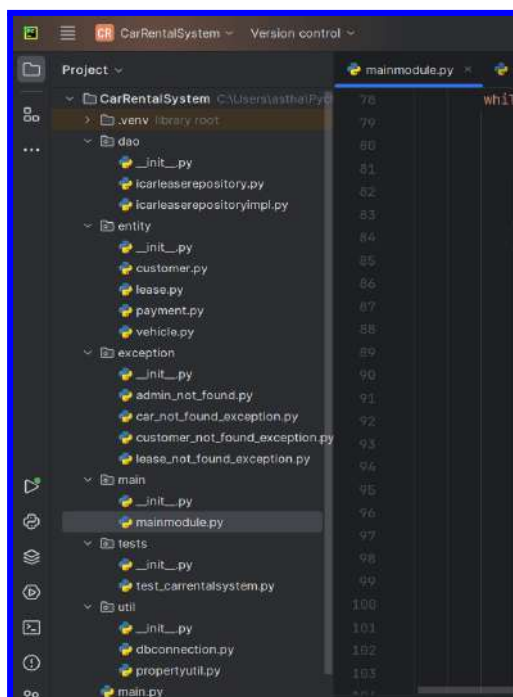
- **util**

- Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.

- Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object(Use method defined in DBPropertyUtil class to get the connection String).

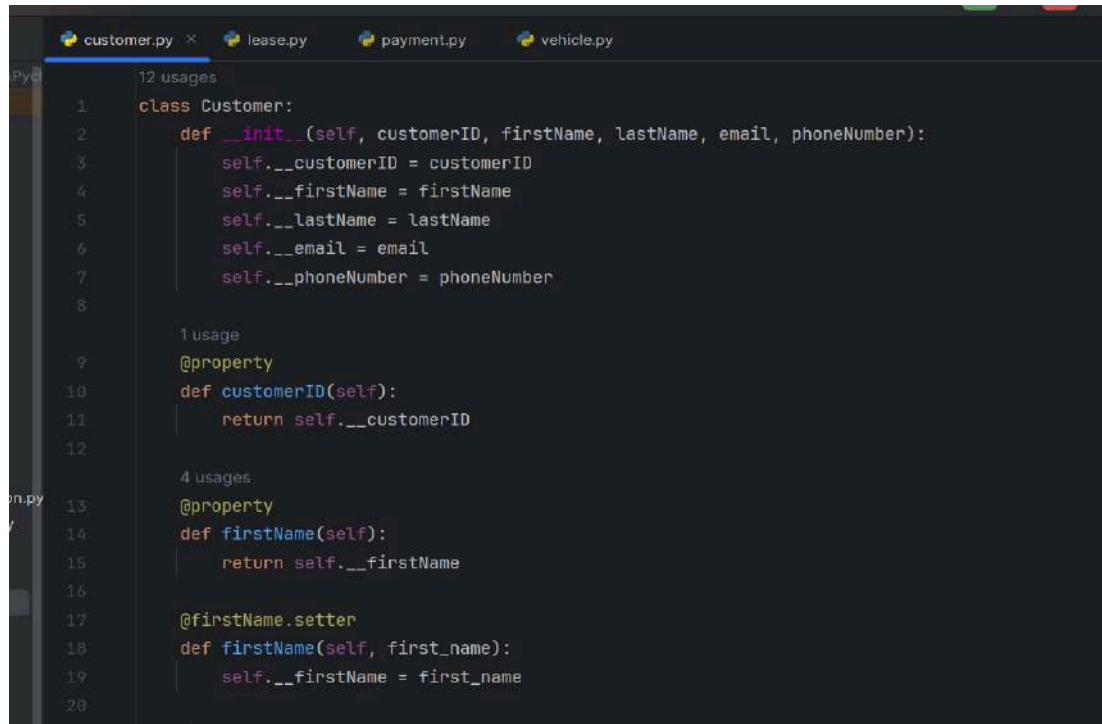
- **main**

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

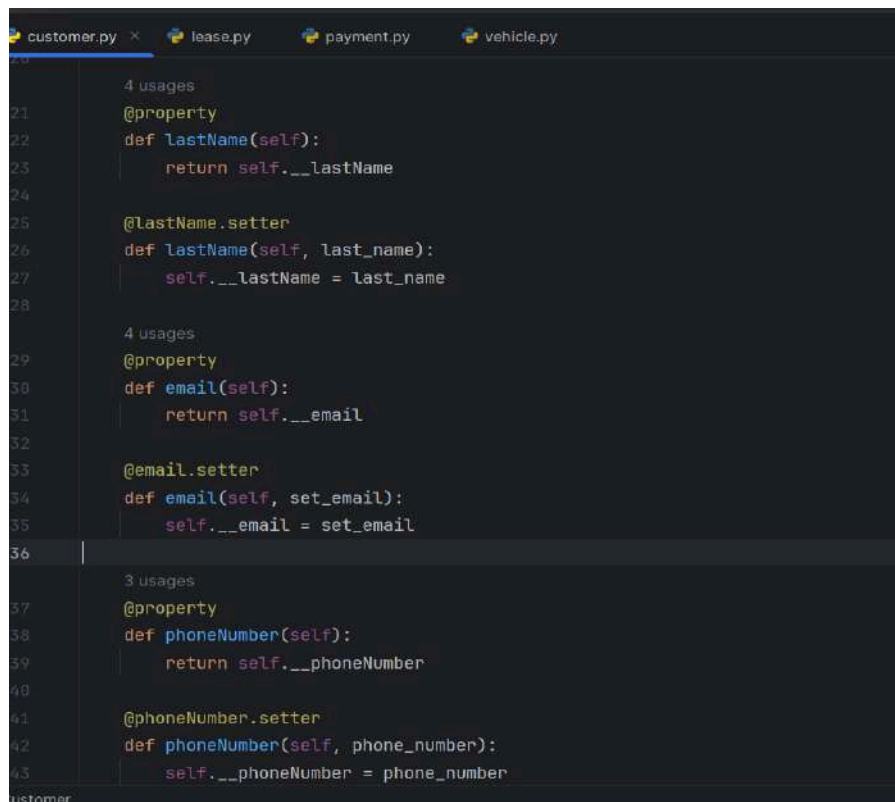


5. Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters, setters)

-> Customer.



```
customer.py x lease.py payment.py vehicle.py
PyCharm
12 usages
1 class Customer:
2     def __init__(self, customerID, firstName, lastName, email, phoneNumber):
3         self.__customerID = customerID
4         self.__firstName = firstName
5         self.__lastName = lastName
6         self.__email = email
7         self.__phoneNumber = phoneNumber
8
9     @property
10    def customerID(self):
11        return self.__customerID
12
13    4 usages
14    @property
15    def firstName(self):
16        return self.__firstName
17
18    @firstName.setter
19    def firstName(self, first_name):
20        self.__firstName = first_name
```



```
customer.py x lease.py payment.py vehicle.py
20
21    4 usages
22    @property
23    def lastName(self):
24        return self.__lastName
25
26    @lastName.setter
27    def lastName(self, last_name):
28        self.__lastName = last_name
29
30    4 usages
31    @property
32    def email(self):
33        return self.__email
34
35    @email.setter
36    def email(self, set_email):
37        self.__email = set_email
38
39    3 usages
40    @property
41    def phoneNumber(self):
42        return self.__phoneNumber
43
44    @phoneNumber.setter
45    def phoneNumber(self, phone_number):
46        self.__phoneNumber = phone_number
customer
```

-> Lease

```
control ▾ mainmodule ▾ ↻
customer.py lease.py × payment.py vehicle.py
18 usages
1 class Lease:
2     def __init__(self, leaseID, vehicleID, customerID, startDate, endDate, type):
3         self.__leaseID = leaseID
4         self.__vehicleID = vehicleID
5         self.__customerID = customerID
6         self.__startDate = startDate
7         self.__endDate = endDate
8         self.__type = type
9
10
11     8 usages
12     @property
13     def leaseID(self):
14         return self.__leaseID
15
16     1 usage
17     @property
18     def vehicleID(self):
19         return self.__vehicleID
20
21     1 usage
22     @property
23     def customerID(self):
24         return self.__customerID
```

```
control ▾
customer.py lease.py × payment.py vehicle.py
23 @property
24 def startDate(self):
25     return self.__startDate
26
27 @startDate.setter
28 def startDate(self, set_startDate):
29     self.__startDate = set_startDate
30
31     3 usages
32     @property
33     def endDate(self):
34         return self.__endDate
35
36     @endDate.setter
37     def endDate(self, set_endDate):
38         self.__endDate = set_endDate
39
40     2 usages
41     @property
42     def type(self):
43         return self.__type
44
45     @type.setter
46     def type(self, set_type):
47         self.__type = set_type
```

->Payment

```
customer.py lease.py payment.py × vehicle.py
5 usages
1 class Payment:
2     def __init__(self, paymentID, leaseID, paymentDate, amount):
3         self.__paymentID = paymentID
4         self.__leaseID = leaseID
5         self.__paymentDate = paymentDate
6         self.__amount = amount
7
8     1 usage
9     @property
10    def paymentID(self):
11        return self.__paymentID
12
13    @property
14    def leaseID(self):
15        return self.__leaseID
16
17    2 usages
18    @property
19    def paymentDate(self):
20        return self.__paymentDate
21
22    @paymentDate.setter
23    def paymentDate(self, payment_date):
24        self.__paymentDate = payment_date
25
26
```

```
customer.py lease.py payment.py × vehicle.py
42
13    @property
14    def leaseID(self):
15        return self.__leaseID
16
17    2 usages
18    @property
19    def paymentDate(self):
20        return self.__paymentDate
21
22    @paymentDate.setter
23    def paymentDate(self, payment_date):
24        self.__paymentDate = payment_date
25
26    2 usages
27    @property
28    def amount(self):
29        return self.__amount
30
31    @amount.setter
32    def amount(self, set_amount):
33        self.__amount = set_amount
34
```


->Vehicle

```
17 usages
1 class Vehicle:
2     def __init__(self, vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity):
3         self.__vehicleID = vehicleID
4         self.__make = make
5         self.__model = model
6         self.__year = year
7         self.__dailyRate = dailyRate
8         self.__status = status
9         self.__passengerCapacity = passengerCapacity
10        self.__engineCapacity = engineCapacity
11
12        4 usages
13        @property
14        def vehicleID(self):
15            return self.__vehicleID
16
17        5 usages
18        @property
19        def make(self):
20            return self.__make
21
22        @make.setter
23        def make(self, set_make):
24            self.__make = set_make
```

```
24        @property
25        def model(self):
26            return self.__model
27
28        @model.setter
29        def model(self, set_model):
30            self.__model = set_model
31
32        2 usages
33        @property
34        def year(self):
35            return self.__year
36
37        @year.setter
38        def year(self, set_year):
39            self.__year = set_year
40
41        2 usages
42        @property
43        def dailyRate(self):
44            return self.__dailyRate
45
46        @dailyRate.setter
47        def dailyRate(self, set_dailyRate):
48            self.__dailyRate = set_dailyRate
```

```
customer.py lease.py payment.py vehicle.py X
48 @property
49 def status(self):
50     return self.__status
51
52 @status.setter
53 def status(self, set_status):
54     self.__status = set_status
55
56 2 usages
57 @property
58 def passengerCapacity(self):
59     return self.__passengerCapacity
60
61 @passengerCapacity.setter
62 def passengerCapacity(self, passenger_capacity):
63     self.__passengerCapacity = passenger_capacity
64
65 2 usages
66 @property
67 def engineCapacity(self):
68     return self.__engineCapacity
69
70 @engineCapacity.setter
71 def engineCapacity(self, engine_capacity):
72     self.__engineCapacity = engine_capacity
```

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. findCarByld(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. findCustomerByld(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

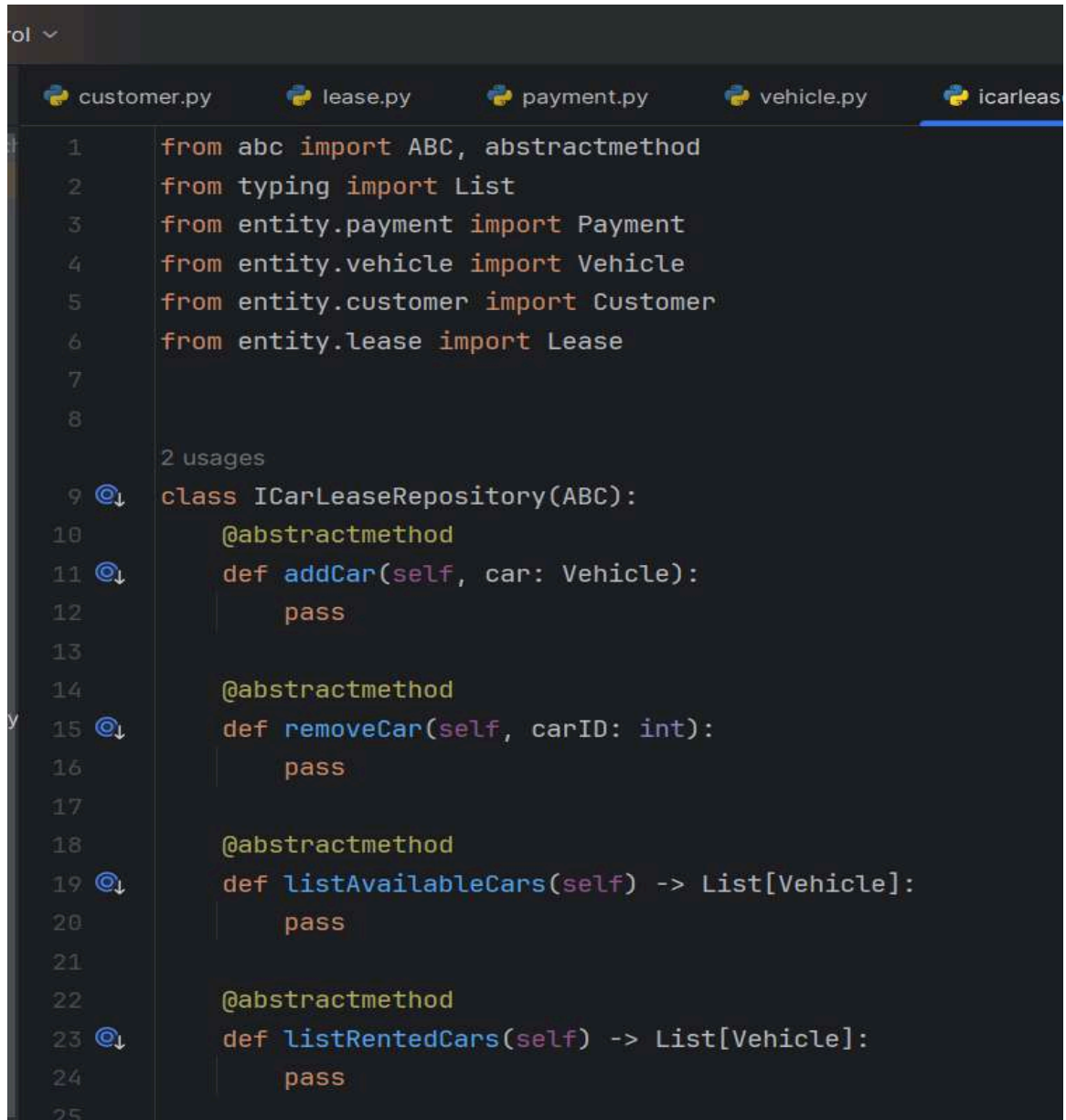
return type : Lease list

- **Payment Handling**

1. **void recordPayment();**

parameter : Lease lease, double amount

return type : void



The screenshot shows a code editor with five tabs: customer.py, lease.py, payment.py, vehicle.py, and icarlease.py. The payment.py tab is active, displaying the following Python code:

```
1 from abc import ABC, abstractmethod
2 from typing import List
3 from entity.payment import Payment
4 from entity.vehicle import Vehicle
5 from entity.customer import Customer
6 from entity.lease import Lease
7
8
9 2 usages
10 class ICarLeaseRepository(ABC):
11     @abstractmethod
12     def addCar(self, car: Vehicle):
13         pass
14
15     @abstractmethod
16     def removeCar(self, carID: int):
17         pass
18
19     @abstractmethod
20     def listAvailableCars(self) -> List[Vehicle]:
21         pass
22
23     @abstractmethod
24     def listRentedCars(self) -> List[Vehicle]:
25         pass
```

```

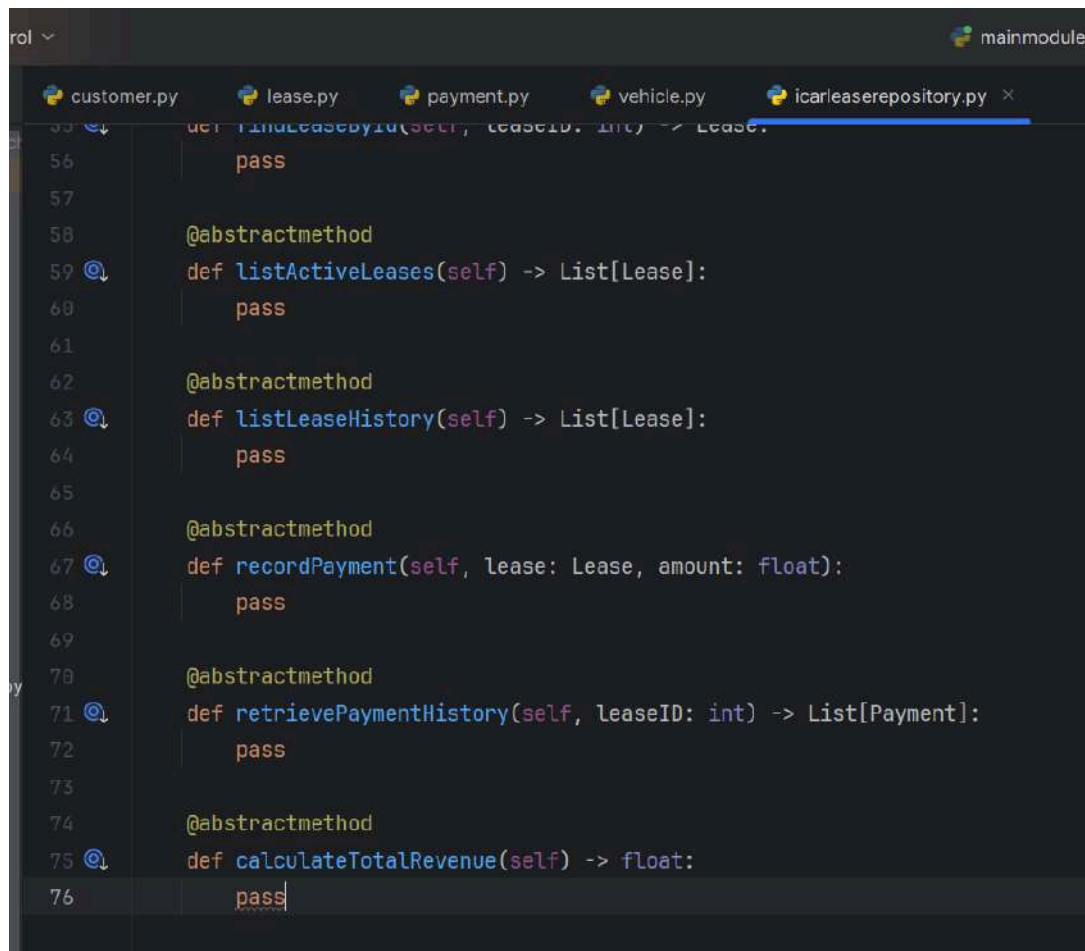
customer.py lease.py payment.py vehicle.py icarleaserepository.py x
24     pass
25
26     @abstractmethod
27     def findCarById(self, carID: int) -> Vehicle:
28         pass
29
30     @abstractmethod
31     def addCustomer(self, customer: Customer):
32         pass
33
34     @abstractmethod
35     def removeCustomer(self, customerID: int):
36         pass
37
38     @abstractmethod
39     def listCustomers(self) -> List[Customer]:
40         pass
41
42     @abstractmethod
43     def findCustomerById(self, customerID: int) -> Customer:
44         pass
45
46     @abstractmethod
47     def createLease(self, customerID: int, carID: int, startDate, endDate, type) -> Lease:
48         pass
49

```

```

control v mainmodule v
customer.py lease.py payment.py vehicle.py icarleaserepository.py x
48     pass
49
50     @abstractmethod
51     def returnCar(self, leaseID: int) -> Lease:
52         pass
53
54     @abstractmethod
55     def findLeaseById(self, leaseID: int) -> Lease:
56         pass
57
58     @abstractmethod
59     def listActiveLeases(self) -> List[Lease]:
60         pass
61
62     @abstractmethod
63     def listLeaseHistory(self) -> List[Lease]:
64         pass
65
66     @abstractmethod
67     def recordPayment(self, lease: Lease, amount: float):
68         pass
69
70     @abstractmethod
71     def retrievePaymentHistory(self, leaseID: int) -> List[Payment]:
72         pass
73

```



The screenshot shows a code editor with a dark theme. The top of the editor has a tab bar with five tabs: 'customer.py', 'lease.py', 'payment.py', 'vehicle.py', and 'icarleaseRepository.py'. The 'icarleaseRepository.py' tab is selected and highlighted in blue. The code in the editor is for an abstract class. It starts with a method 'findLeaseById' on line 55, followed by a 'pass' statement on line 56. Then, there are three abstract methods: 'listActiveLeases' on line 59, 'listLeaseHistory' on line 63, and 'recordPayment' on line 67. Each of these is followed by a 'pass' statement. Finally, there are two more abstract methods: 'retrievePaymentHistory' on line 71 and 'calculateTotalRevenue' on line 75, each followed by a 'pass' statement on line 76. The line numbers 55 through 76 are visible on the left side of the editor. The code is as follows:

```
55 def findLeaseById(self, leaseID: int) -> Lease:
56     pass
57
58     @abstractmethod
59     def listActiveLeases(self) -> List[Lease]:
60         pass
61
62     @abstractmethod
63     def listLeaseHistory(self) -> List[Lease]:
64         pass
65
66     @abstractmethod
67     def recordPayment(self, lease: Lease, amount: float):
68         pass
69
70     @abstractmethod
71     def retrievePaymentHistory(self, leaseID: int) -> List[Payment]:
72         pass
73
74     @abstractmethod
75     def calculateTotalRevenue(self) -> float:
76         pass
```

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

```
4 usages
14 class ICarLeaseRepositoryImpl(ICarLeaseRepository):
15     def __init__(self, connection=None):
16         self.connection = connection
17
18     2 usages
19     def addCar(self, car: Vehicle):
20         cursor = self.connection.cursor()
21         try:
22             cursor.execute(
23                 "INSERT INTO vehicle (make, model, year, dailyRate, status, passengerCapacity, engineCapacity) "
24                 "VALUES (%s, %s, %s, %s, %s, %s, %s)",
25                 (car.make, car.model, car.year, car.dailyRate, car.status,
26                  car.passengerCapacity, car.engineCapacity))
27             self.connection.commit()
28         finally:
29             cursor.close()
30
31     1 usage
32     def removeCar(self, carID: int):
33         cursor = self.connection.cursor()
34         try:
35             # Check if the car has active leases
36             active_leases = self.listActiveLeases()
37             active_leases_for_car = [lease for lease in active_leases if lease.vehicleID == carID]
```

```
customer.py lease.py payment.py vehicle.py icarleaseRepository.py icarleaseRepositoryImpl.py x
35     active_leases_for_car = [lease for lease in active_leases if lease.vehicleID == carID]
36
37     if active_leases_for_car:
38         raise ActiveLeasesExistException("Cannot remove car with active leases.")
39
40     cursor.execute("DELETE FROM vehicle WHERE vehicleID=%s", (carID,))
41     if cursor.rowcount == 0:
42         raise CarNotFoundException(f"Car with ID {carID} not found.")
43     self.connection.commit()
44 finally:
45     cursor.close()
46
47     1 usage
48     def listAvailableCars(self) -> List[Vehicle]:
49         cursor = self.connection.cursor()
50         try:
51             cursor.execute("SELECT * FROM vehicle WHERE status='available'")
52             rows = cursor.fetchall()
53
54             cars = []
55             for row in rows:
56                 car = Vehicle(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
57                 cars.append(car)
58
59             return cars
60         finally:
61             cursor.close()
```



```
mainmodule
customer.py lease.py payment.py vehicle.py icarleaserepository.py icarleaserepositoryimpl.py

61
1 usage
62 def listRentedCars(self) -> List[Vehicle]:
63     cursor = self.connection.cursor()
64     try:
65         cursor.execute("SELECT * FROM vehicle WHERE status='notAvailable'")
66         rows = cursor.fetchall()
67
68         cars = []
69         for row in rows:
70             car = Vehicle(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
71             cars.append(car)
72
73         return cars
74     finally:
75         cursor.close()
```

```
mainmodule
customer.py lease.py payment.py vehicle.py icarleaserepository.py icarleaserepositoryimpl.py

77 def findCarById(self, carID: int) -> Vehicle:
78     cursor = self.connection.cursor()
79     try:
80         cursor.execute("SELECT * FROM vehicle WHERE vehicleID=%s", (carID,))
81         row = cursor.fetchone()
82
83         if row:
84             return Vehicle(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
85         else:
86             raise CarNotFoundException(f"Car with ID {carID} not found.")
87     finally:
88         cursor.close()
89
1 usage
90 def addCustomer(self, customer: Customer):
91     cursor = self.connection.cursor()
92     try:
93         cursor.execute("INSERT INTO customer (firstName, lastName, email, phoneNumber) VALUES (%s, %s, %s, %s)",
94             (customer.firstName, customer.lastName, customer.email, customer.phoneNumber))
95         self.connection.commit()
96     finally:
97         cursor.close()
98
```



```
1 usage
99 @
def removeCustomer(self, customerID: int):
100     cursor = self.connection.cursor()
101     try:
102         # Check if the customer has active leases
103         active_leases = self.listActiveLeases()
104         active_leases_for_customer = [lease for lease in active_leases if lease.customerID == customerID]
105
106         if active_leases_for_customer:
107             raise ActiveLeasesExistException("Cannot remove customer with active leases.")
108
109         cursor.execute("DELETE FROM customer WHERE customerID=%s", (customerID,))
110         self.connection.commit()
111     finally:
112         cursor.close()
113
```

```
customer.py lease.py payment.py vehicle.py icarleaseRepository.py icarleaseRepositoryImpl.py
110 @
def listCustomers(self) -> List[Customer]:
111     cursor = self.connection.cursor()
112     try:
113         cursor.execute("SELECT * FROM customer")
114         rows = cursor.fetchall()
115
116         customers = []
117         for row in rows:
118             customer = Customer(row[0], row[1], row[2], row[3], row[4])
119             customers.append(customer)
120
121         return customers
122     finally:
123         cursor.close()
124
125 3 usages
@
def findCustomerById(self, customerID: int) -> Customer:
126     cursor = self.connection.cursor()
127     try:
128         cursor.execute("SELECT * FROM customer WHERE customerID=%s", (customerID,))
129         row = cursor.fetchone()
130
131         if row:
132             return Customer(row[0], row[1], row[2], row[3], row[4])
133         else:
134             raise CustomerNotFoundException(f"Customer with ID {customerID} not found.")
135
136     ...
ICarLeaseRepositoryImpl | update_customer_info() | try | If email is not None
```



```
bl ▾ mainmodule ▾ ↺ ⚙ □ ⋮
customer.py lease.py payment.py vehicle.py icarleaserepository.py icarleaserepositoryimpl.py
116         try:
117             cursor.execute("SELECT * FROM customer")
118             rows = cursor.fetchall()
119
120             customers = []
121             for row in rows:
122                 customer = Customer(row[0], row[1], row[2], row[3], row[4])
123                 customers.append(customer)
124
125             return customers
126         finally:
127             cursor.close()
128
129 3 usages
129 ② def findCustomerById(self, customerID: int) -> Customer:
130     cursor = self.connection.cursor()
131     try:
132         cursor.execute("SELECT * FROM customer WHERE customerID=%s", (customerID,))
133         row = cursor.fetchone()
134
135         if row:
136             return Customer(row[0], row[1], row[2], row[3], row[4])
137         else:
138             raise CustomerNotFoundException(f"Customer with ID {customerID} not found.")
139     finally:
140         cursor.close()
141
```



```
3 usages
181 def listActiveLeases(self) -> List[Lease]:
182     cursor = self.connection.cursor()
183     try:
184         cursor.execute("SELECT * FROM lease WHERE startDate <= CURDATE() AND endDate >= CURDATE()")
185         rows = cursor.fetchall()
186         leases = []
187         for row in rows:
188             lease = Lease(row[0], row[1], row[2], row[3], row[4], row[5])
189             leases.append(lease)
190
191         return leases
192     finally:
193         cursor.close()
194
1 usage
195 def listLeaseHistory(self) -> List[Lease]:
196     if not self.connection:
197         raise ValueError("Connection is not provided.")
198     cursor = self.connection.cursor()
199     try:
200         cursor.execute("SELECT * FROM lease")
201         rows = cursor.fetchall()
202
203         leases = []
204         for row in rows:
205             lease = Lease(row[0], row[1], row[2], row[3], row[4], row[5])
206
ICarLeaseRepositoryImpl
```

```
1 usage
11 def recordPayment(self, lease: Lease, amount: float):
12     cursor = self.connection.cursor()
13     try:
14         print("Lease id", lease.leaseID)
15         cursor.execute("INSERT INTO payment (leaseID, paymentDate, amount) VALUES (%s, %s, %s)",
16                         (lease.leaseID, datetime.now().strftime("%Y-%m-%d"), amount))
17         self.connection.commit()
18     finally:
19         cursor.close()
20
1 usage
21 def retrievePaymentHistory(self, leaseID: int) -> List[Payment]:
22     cursor = self.connection.cursor()
23     try:
24         cursor.execute("SELECT * FROM Payment WHERE leaseID = %s", (leaseID,))
25         rows = cursor.fetchall()
26         payment_history = []
27         for row in rows:
28             payment = Payment(row[0], row[1], row[2], row[3])
29             payment_history.append(payment)
30         self.connection.commit()
31         return payment_history
32     except Exception as e:
33         raise LeaseNotFoundException(f"Lease with ID {leaseID} not found.") from e
ICarLeaseRepositoryImpl
```



```

1 usage
237 @
238     def calculateTotalRevenue(self) -> float:
239         cursor = self.connection.cursor()
240         try:
241             cursor.execute("SELECT SUM(amount) FROM Payment")
242             total_revenue = cursor.fetchone()[0]
243             return total_revenue
244         finally:
245             cursor.close()

```

|CarLeaseRepositoryImpl|

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 file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```

nicle.py  icarleaseRepository.py  test_carrentalsystem.py  customer.py  lease.py  dbconnection.py
1  import mysql.connector
2  from mysql.connector import Error
3  from util.propertyutil import propertyUtil
4
5
6  9 usages
7  class DBConnection:
8
9      connection = None
10
11  2 usages
12  @staticmethod
13  def getConnection():
14      if DBConnection.connection is None:
15          try:
16              connection_string = propertyUtil.getPropertyString()
17              DBConnection.connection = mysql.connector.connect(**connection_string)
18
19              if DBConnection.connection.is_connected():
20                  print("Database connected successfully")
21          except Error as e:
22              print(f'Error : {e}')
23
24      return DBConnection.connection

```

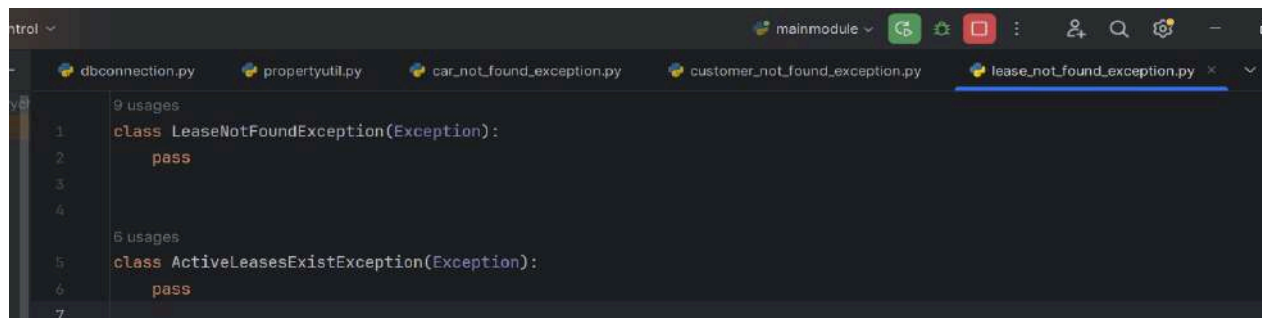
```
lease.py  payment.py  vehicle.py  icarleaserepository.py  icarleaserepositoryimpl.py
1 2 usages
2 class propertyUtil:
3
4     1 usage
5     @staticmethod
6     def getPropertyString():
7         connection_string = {
8             'host': "localhost",
9             'database': "carrentalsystem",
10            'user': "root",
11            'password': "ABcd300#$"
12        }
13
14     return connection_string
```

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.

```
control  mainmodule
vehicle.py  icarleaserepository.py  icarleaserepositoryimpl.py  dbconnection.py  propertyutil.py  car_not_found_exception.py
1 9 usages
2 class CarNotFoundException(Exception):
3     pass
4
```

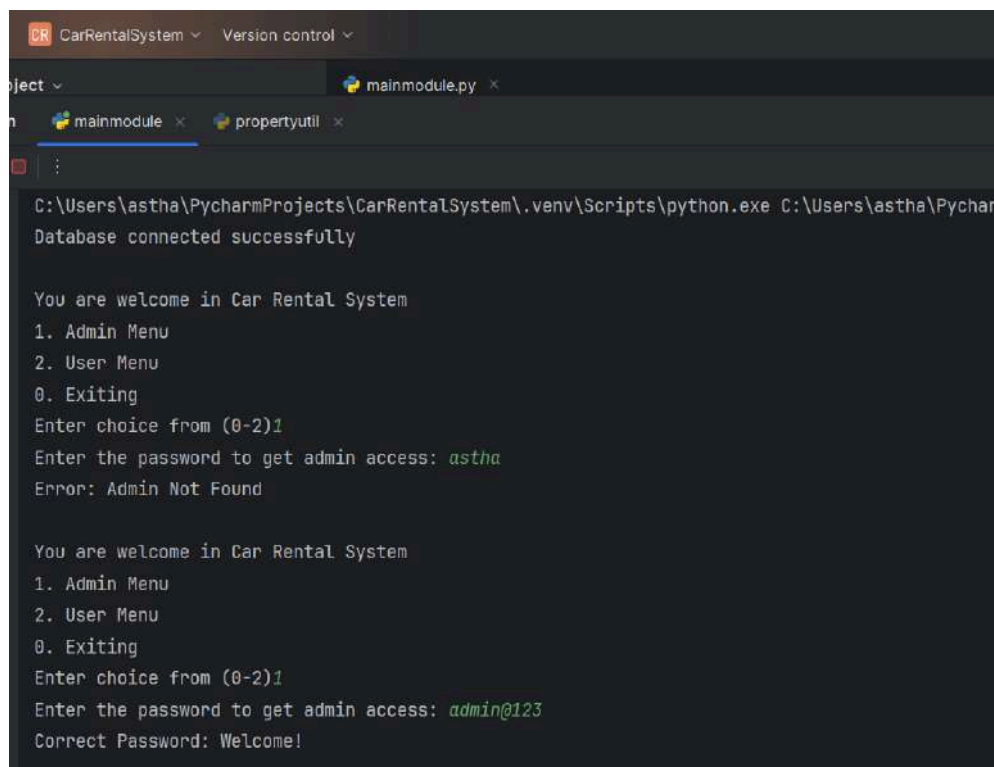
```
control  mainmodule
icarleaserepositoryimpl.py  dbconnection.py  propertyutil.py  car_not_found_exception.py  customer_not_found_exception.py
1 9 usages
2 class CustomerNotFoundException(Exception):
3     pass
```



```
9 usages
1 class LeaseNotFoundException(Exception):
2     pass
3
4
5 6 usages
6 class ActiveLeasesExistException(Exception):
7     pass
```

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

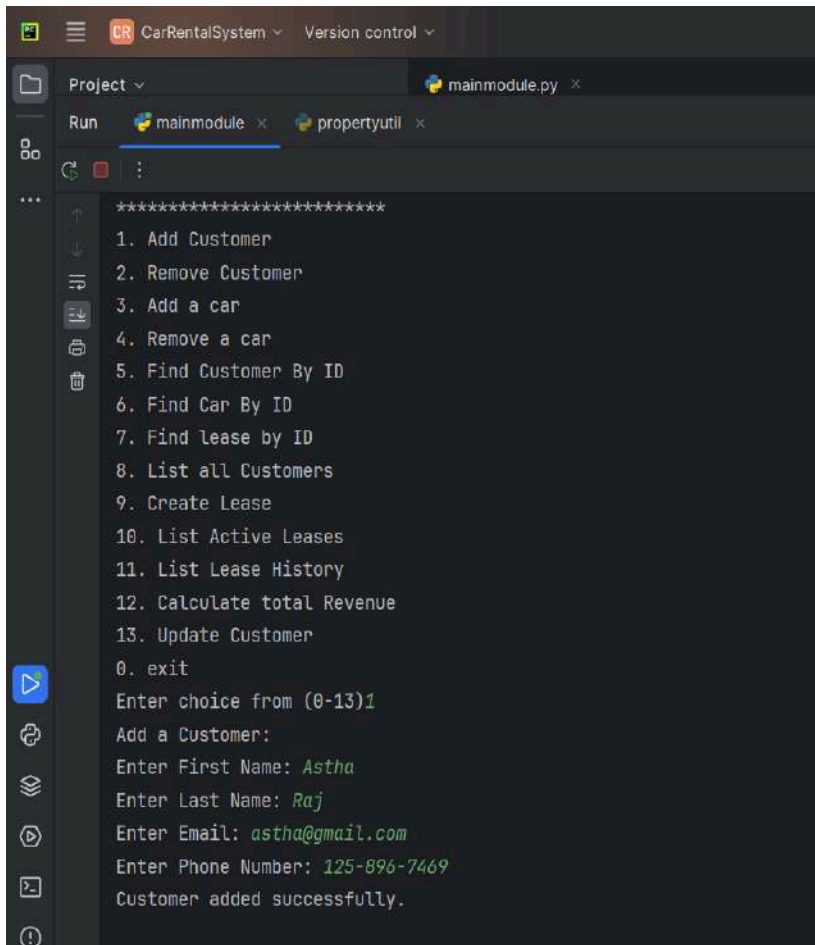
-> OUTPUTS OF CAR RENTAL SYSTEM



```
CR CarRentalSystem Version control
mainmodule.py
mainmodule propertyutil
C:\Users\astha\PycharmProjects\CarRentalSystem\.venv\Scripts\python.exe C:\Users\astha\Pychar
Database connected successfully

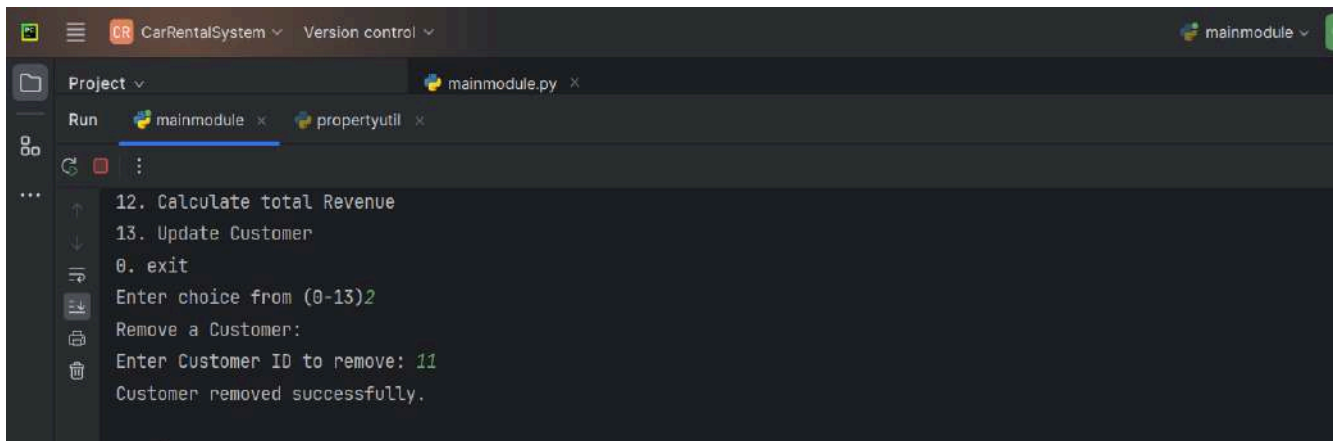
You are welcome in Car Rental System
1. Admin Menu
2. User Menu
0. Exiting
Enter choice from (0-2)1
Enter the password to get admin access: astha
Error: Admin Not Found

You are welcome in Car Rental System
1. Admin Menu
2. User Menu
0. Exiting
Enter choice from (0-2)1
Enter the password to get admin access: admin@123
Correct Password: Welcome!
```

The screenshot shows a code editor with a project named 'CarRentalSystem'. The 'Run' tab is active, displaying the output of a Python script. The script presents a menu of 13 options, with '0. exit' at the bottom. The user has entered '1' to select '1. Add Customer'. The script then prompts for the first name, last name, email, and phone number, which the user has entered as 'Astha', 'Raj', 'astha@gmail.com', and '125-896-7469' respectively. The final output is 'Customer added successfully.'.

```
*****
1. Add Customer
2. Remove Customer
3. Add a car
4. Remove a car
5. Find Customer By ID
6. Find Car By ID
7. Find lease by ID
8. List all Customers
9. Create Lease
10. List Active Leases
11. List Lease History
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)1
Add a Customer:
Enter First Name: Astha
Enter Last Name: Raj
Enter Email: astha@gmail.com
Enter Phone Number: 125-896-7469
Customer added successfully.
```



The screenshot shows the same code editor with the 'Run' tab active. The menu is displayed again, and the user has entered '2' to select '2. Remove Customer'. The script prompts for the customer ID to remove, which the user has entered as '11'. The final output is 'Customer removed successfully.'.

```
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)2
Remove a Customer:
Enter Customer ID to remove: 11
Customer removed successfully.
```

```
...
3. Add a car
4. Remove a car
5. Find Customer By ID
6. Find Car By ID
7. Find lease by ID
8. List all Customers
9. Create Lease
10. List Active Leases
11. List Lease History
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)3
Adding a new Car:
Enter Make: Audi
Enter Model: A5
Enter Year: 2023
Enter Daily Rate: 55.00
Enter Status (Available/NotAvailable): NotAvailable
Enter Passenger Capacity: 4
Enter Engine Capacity: 3500
Great! Car added successfully. None
```

```
13. Update Customer
0. exit
Enter choice from (0-13)4
Remove a Car:
Enter Car ID to remove: 11
Car removed successfully.
```

```
Project mainmodule.py
Run mainmodule x propertyutil x
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)4
Remove a Car:
Enter Car ID to remove: 16
Error: Car with ID 16 not found.
Admin Menu
```

```
CarRentalSystem Version control
Project mainmodule.py
Run mainmodule x propertyutil x
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)5
Find Customer by ID:
Enter Customer ID to find: 10
Customer Found: Olivia Adams - Email: olivia@example.com
Admin Menu
*****
```

```
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)6
Find Car by ID:
Enter Car ID to find: 5
Car Found: Chevrolet Malibu - Status: Available

Admin Menu
*****
1. Add Customer
2. Remove Customer
```

```
Project CarRentalSystem Version control
mainmodule.py
Run mainmodule x propertyutil x
13. Update Customer
0. exit
Enter choice from (0-13)7
Finding lease by id
Enter lease ID: 10
Lease found:
Lease ID:,10, lease-type: MonthlyLease

Admin Menu
*****
```

```
0. exit
Enter choice from (0-13)8
List Customers:
Customers:
1. John Doe - Email: johndoe@example.com, PhoneNumber: 555-555-5555
2. Jane Smith - Email: janesmith@example.com, PhoneNumber: 555-123-4567
3. Robert Johnson - Email: robert@example.com, PhoneNumber: 555-789-1234
4. Sarah Brown - Email: sarah@example.com, PhoneNumber: 555-456-7890
5. David Lee - Email: david@example.com, PhoneNumber: 555-987-6543
6. Laura Hall - Email: laura@example.com, PhoneNumber: 555-234-5678
7. Michael Davis - Email: michael@example.com, PhoneNumber: 555-876-5432
8. Emma Wilson - Email: emma@example.com, PhoneNumber: 555-432-1098
9. William Taylor - Email: william@example.com, PhoneNumber: 555-321-6547
10. Olivia Adams - Email: olivia@example.com, PhoneNumber: 123-896-7458
12. Astha Raj - Email: astha@gmail.com, PhoneNumber: 125-896-7469

Admin Menu
```

```
13. Update Customer
0. exit
Enter choice from (0-13)9
Creating Lease:
Enter Customer ID: 3
Enter Vehicle ID: 3
Enter Start Date (YYYY-MM-DD): 2024-02-02
Enter End Date (YYYY-MM-DD): 2024-03-02
Input type as DailyLease or MonthlyLease: MonthlyLease
Lease created successfully. Lease ID: 11

Admin Menu
```

```
7. Create Lease
10. List Active Leases
11. List Lease History
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)10
List Active Leases:
Active Leases:
Lease ID: 11 - Start Date: 2024-02-02 - End Date: 2024-03-02

Admin Menu
*****
1. Add Customer
2. Remove Customer
3. Add a car
4. Remove a car
5. Find Customer By ID
```

```
11. List Lease History
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)11
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: 2024-02-02 - End Date: 2024-03-02

Admin Menu
```

```
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)12
Total Revenue: 6155.00

Admin Menu
```

```
11. List Lease History
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)13
Enter Customer ID: 10
Enter new Email (leave blank to keep unchanged):
Enter new Phone (leave blank to keep unchanged): 125-859-7469
Customer information updated successfully.
```

```
Admin Menu
*****
1. Add Customer
2. Remove Customer
3. Add a car
4. Remove a car
5. Find Customer By ID
6. Find Car By ID
7. Find lease by ID
8. List all Customers
9. Create Lease
10. List Active Leases
11. List Lease History
12. Calculate total Revenue
13. Update Customer
0. exit
Enter choice from (0-13)0
Exiting Car Rental System. Goodbye!

You are welcome in Car Rental System
1. Admin Menu
2. User Menu
0. Exiting
Enter choice from (0-2)
```

```
You are welcome in Car Rental System
1. Admin Menu
2. User Menu
0. Exiting
Enter choice from (0-2)2
Enter username: s
Enter password:
Invalid username or password. Please try again.
Enter username: astha
Enter password: raj
User login successful!

User Menu
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
Enter choice from (0-4)|
```

```
User Menu
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
Enter choice from (0-4)1
List Rented Cars:
Rented Cars:
3. Ford Focus - Status: NotAvailable
6. Hyundai Sonata - Status: NotAvailable
9. Audi A4 - Status: NotAvailable

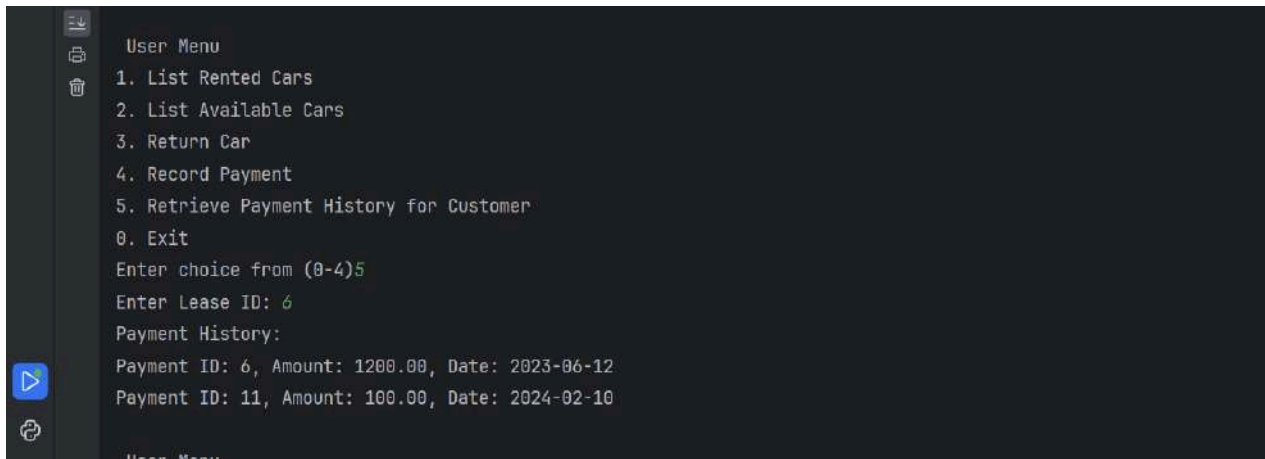
User Menu
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
```



```
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
Enter choice from (0-4)2
List Available Cars:
Available Cars:
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
```

```
User Menu
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
Enter choice from (0-4)3
Return Car:
Enter Lease ID to return the car: 9
Car returned successfully. Lease ID: 9
```

```
User Menu
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
Enter choice from (0-4)4
Record Payment:
Enter Lease ID for payment: 6
Enter Payment Amount: 100
Lease id 6
Payment recorded successfully.
```

A screenshot of a terminal window with a dark background. On the left side, there is a vertical toolbar with icons for file operations (new, open, save, delete) and a play button. The terminal text is as follows:

```
User Menu
1. List Rented Cars
2. List Available Cars
3. Return Car
4. Record Payment
5. Retrieve Payment History for Customer
0. Exit
Enter choice from (0-4)5
Enter Lease ID: 6
Payment History:
Payment ID: 6, Amount: 1200.00, Date: 2023-06-12
Payment ID: 11, Amount: 100.00, Date: 2024-02-10
User Menu
```

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 tests in test_carrentalsystem.py
propertyutil.py car_not_found_exception.py customer_not_found_exception.py lease_not_found_exception.py test_carrentalsystem.py x

11
12 @pytest.fixture
13 def car_repository():
14     connection = DBConnection.getConnection()
15     return ICarLeaseRepositoryImpl(connection)
16
car_repository()

on tests in test_carrentalsystem.py x
:
:
ms: ✓ Tests passed: 6 of 6 tests - 11 ms

C:\Users\astha\PycharmProjects\CarRentalSystem\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition
Testing started at 23:41 ...
Launching pytest with arguments C:\Users\astha\PycharmProjects\CarRentalSystem\tests\test_carrentalsystem.py --no-header --no-

===== test session starts =====
collecting ... collected 6 items

test_carrentalsystem.py::test_add_car Database connected successfully
PASSED [ 16%]
test_carrentalsystem.py::test_create_lease PASSED [ 33%]
test_carrentalsystem.py::test_retrieve_lease PASSED [ 50%]
test_carrentalsystem.py::test_exception_for_customer_not_found PASSED [ 66%]
test_carrentalsystem.py::test_exception_for_car_not_found PASSED [ 83%]
test_carrentalsystem.py::test_exception_for_lease_not_found PASSED [100%]

===== 6 passed in 0.18s =====
```

```
Python tests in test_carrentalsystem.py
propertyutil.py car_not_found_exception.py customer_not_found_exception.py lease_not_found_exc

1 import pytest
2 from main.mainmodule import MainModule
3 from dao.icarleaserepositoryimpl import ICarLeaseRepositoryImpl
4 from entity.vehicle import Vehicle
5 from exception.car_not_found_exception import CarNotFoundExpection
6 from exception.customer_not_found_exception import CustomerNotFoundExpection
7 from exception.lease_not_found_exception import LeaseNotFoundExpection
8 import datetime
9 from util.dbconnection import DBConnection
10
11
12 @pytest.fixture
13 def car_repository():
14     connection = DBConnection.getConnection()
15     return ICarLeaseRepositoryImpl(connection)
16
17
18 @pytest.fixture
19 def main_module(car_repository):
20     return MainModule()
21
22
23 def test_add_car(main_module, car_repository):
24     car_id = 1
25     make = "Toyota"
26     model = "Camry"
27     year = 2022
28
car_repository()
```

```
Python tests in test_carrentalsystem.py
propertyutil.py car_not_found_exception.py customer_not_found_exception.py lease_not_found_exception.py test_ca

23 > def test_add_car(main_module, car_repository):
24     car_id = 1
25     make = "Toyota"
26     model = "Camry"
27     year = 2022
28     daily_rate = 30
29     status = "available"
30     passenger_capacity = 5
31     engine_capacity = 3
32
33     new_car = Vehicle(car_id, make, model, year, daily_rate, status, passenger_capacity, engine_capacity)
34     car_repository.addCar(new_car)
35
36     assert car_repository.findCarById(new_car.vehicleID)
37
38
39 > def test_create_lease(main_module, car_repository):
40     customer_id = 1
41     car_id = 1
42     start_date = datetime.datetime.now().strftime("%Y-%m-%d")
43     end_date = "2024-02-07"
44     lease_type = "Monthly"
45
46     created_lease = car_repository.createLease(customer_id, car_id, start_date, end_date, lease_type)
47
48     assert car_repository.findLeaseById(created_lease.leaseID)
49
50
51 > def test_retrieve_lease(main_module, car_repository):
car_repository()
```

```
Python tests in test_carrentalsystem.py
propertyutil.py car_not_found_exception.py customer_not_found_exception.py lease_not_found_exception.py test

44     lease_type = "Monthly"
45
46     created_lease = car_repository.createLease(customer_id, car_id, start_date, end_date, lease_type)
47
48     assert car_repository.findLeaseById(created_lease.leaseID)
49
50
51 > def test_retrieve_lease(main_module, car_repository):
52     lease_id = 1
53     retrieved_lease = car_repository.findLeaseById(lease_id)
54     assert retrieved_lease is not None
55
56
57 > def test_exception_for_customer_not_found(main_module, car_repository):
58     with pytest.raises(CustomerNotFoundException):
59         car_repository.findCustomerById(9999)
60
61
62 > def test_exception_for_car_not_found(main_module, car_repository):
63     with pytest.raises(CarNotFoundException):
64         car_repository.findCarById(9999)
65
66
67 > def test_exception_for_lease_not_found(main_module, car_repository):
68     with pytest.raises(LeaseNotFoundException):
69         car_repository.findLeaseById(9999)
70
```

```
CarRentalSystem - Version control
mainmodule.py test_carrentalsystem.py propertyutil.py carleaserepository.py dbconnection.py lease_not_found_exception.py
Terminal Local x + -
(.venv) PS C:\Users\astha\PycharmProjects\CarRentalSystem> pytest --h 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 --h 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 --h 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