

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	1	2023-01-01	2023-01-05	Daily
2	2	2	2	2023-02-15	2023-02-28	Monthly
3	3	3	3	2023-03-10	2023-03-15	Daily
4	4	4	4	2023-04-20	2023-04-30	Monthly
5	5	5	5	2023-05-05	2023-05-10	Daily
6	6	4	3	2023-06-15	2023-06-30	Monthly
7	7	7	7	2023-07-01	2023-07-10	Daily
8	8	8	8	2023-08-12	2023-08-15	Monthly
9	9	3	3	2023-09-07	2023-09-10	Daily
10	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)

```

Project: CarRentalSystem
├── CarRentalSystem
│   ├── __init__.py
│   ├── carlease.py
│   ├── carleaseimpl.py
│   ├── entity
│   │   ├── __init__.py
│   │   ├── customer.py
│   │   ├── lease.py
│   │   ├── payment.py
│   │   └── vehicle.py
│   ├── exception
│   │   ├── __init__.py
│   │   ├── car_not_found_exception.py
│   │   ├── customer_not_found_exception.py
│   │   └── lease_not_found_exception.py
│   ├── main
│   │   ├── __init__.py
│   │   └── mainmodule.py
│   ├── tests
│   │   ├── __init__.py
│   │   └── test_carrentalsystem.py
│   └── util
│       ├── __init__.py
│       ├── db.properties
│       ├── dbconnection.py
│       ├── propertyutil.py
│       └── main.py
└── main.py

mainmodule.py
test_carrentalsystem.py
customer.py
dbconnection.py
carleaseimpl.py
payment.py
vehicle.py

13 usages
class Customer:
2
3     def __init__(self, customerID, firstName, lastName, email, phoneNumber):
4         self.__customerID = customerID
5         self.__firstName = firstName
6         self.__lastName = lastName
7         self.__email = email
8         self.__phoneNumber = phoneNumber
9
10    1 usage
11    @property
12    def customerID(self):
13        return self.__customerID
14
15    3 usages
16    @property
17    def firstName(self):
18        return self.__firstName
19
20    3 usages
21    @property
22    def lastName(self):
23        return self.__lastName
24
25    3 usages
26    @property
27    def email(self):
28        return self.__email
29
30    Customer = phoneNumber()

```

```
10 usages
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

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

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

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

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

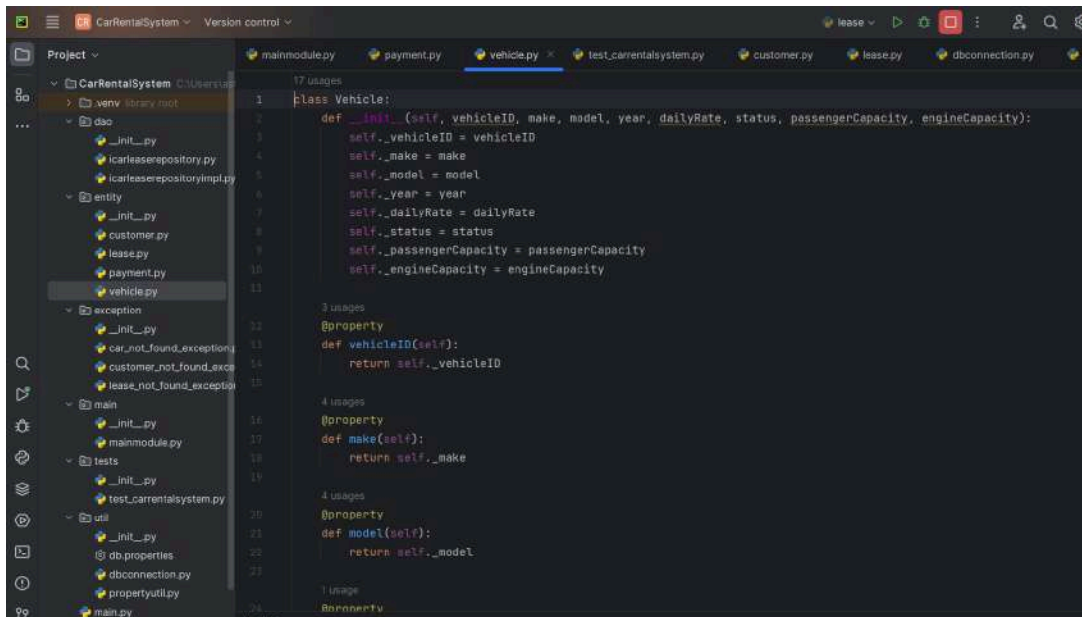
```
2 usages
class Payment:
    def __init__(self, paymentID, leaseID, paymentDate, amount):
        self._paymentID = paymentID
        self._leaseID = leaseID
        self._paymentDate = paymentDate
        self._amount = amount

    @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. 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. 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

The screenshot shows an IDE window for a project named 'CarRentalSystem'. The file explorer on the left shows the project structure, including directories like 'entity', 'exception', 'main', 'tests', and 'util'. The main editor displays the file 'icarleaseRepository.py'. The code defines an abstract class 'ICARLeaseRepository' with several abstract methods. The implementation class 'ICARLeaseRepositoryImpl' is also visible in the file explorer.

```
1 from abc import ABC, abstractmethod
2 from typing import List
3 from entity.Vehicle import Vehicle
4 from entity.customer import Customer
5 from entity.Lease import Lease
6
7
8 class ICARLeaseRepository(ABC):
9     @abstractmethod
10     def addCar(self, car: Vehicle):
11         pass
12
13     @abstractmethod
14     def removeCar(self, carID: int):
15         pass
16
17     @abstractmethod
18     def listAvailableCars(self) -> List[Vehicle]:
19         pass
20
21     @abstractmethod
22     def listRentedCars(self) -> List[Vehicle]:
23         pass
24
25     @abstractmethod
26     def findCarById(self, carID: int) -> Vehicle:
27         pass
28
29     @abstractmethod
```

The screenshot shows the same IDE window, but the code in 'icarleaseRepository.py' is scrolled down to show the implementation of the 'ICARLeaseRepository' class. The code defines several abstract methods and their implementations. The implementation class 'ICARLeaseRepositoryImpl' is also visible in the file explorer.

```
27
28
29     @abstractmethod
30     def addCustomer(self, customer: Customer):
31         pass
32
33     @abstractmethod
34     def removeCustomer(self, customerID: int):
35         pass
36
37     @abstractmethod
38     def listCustomers(self) -> List[Customer]:
39         pass
40
41     @abstractmethod
42     def findCustomerById(self, customerID: int) -> Customer:
43         pass
44
45     @abstractmethod
46     def createLease(self, customerID: int, carID: int, startDate, endDate, type) -> Lease:
47         pass
48
49     @abstractmethod
50     def returnCar(self, leaseID: int) -> Lease:
51         pass
52
53     @abstractmethod
54     def findLeaseById(self, leaseID: int) -> Lease:
55         pass
56
57
```


The screenshot shows an IDE with the project 'CarRentalSystem'. The file explorer on the left shows the package structure: dao, entity, exception, main, tests, and util. The main editor displays the 'icarleaserepository.py' file, which defines an abstract class 'ICarLeaseRepository' with the following methods:

```
class ICarLeaseRepository:
    @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.

The screenshot shows the same IDE with the file 'icarleaserepositoryimpl.py' open in the main editor. The file contains the implementation of the 'ICarLeaseRepository' interface. It starts with a series of imports from various modules, followed by the class definition:

```
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 CarNotFoundExpection
from exception.customer_not_found_exception import CustomerNotFoundExpection
from exception.lease_not_found_exception import LeaseNotFoundExpection
from util.dbconnection import DBConnection
from util.propertyutil import propertyUtil
from typing import List
from datetime import datetime

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

    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, car.passengerCapacity, car.engineCapacity)
            )
```


The screenshot shows an IDE with the project 'CarRentalSystem' open. The file explorer on the left shows the project structure, including packages like dao, entity, exception, main, tests, and util. The main editor displays the file 'icarleaseRepositoryImpl.py'. The code defines a class 'IcarLeaseRepositoryImpl' with a method 'listCustomers' that returns a list of 'Customer' objects. The method uses a database cursor to execute a 'SELECT * FROM customer' query and fetches all rows. Each row is converted into a 'Customer' object using the 'Customer' constructor, and the objects are appended to a list. The cursor is closed at the end of the method. The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces, and the Python version is 3.12.

```
184 cursor = self.connection.cursor()
185
186 try:
187     cursor.execute("DELETE FROM customer WHERE customerID=%s", (customerID,))
188     self.connection.commit()
189 finally:
190     cursor.close()
191
192 1 usage
193 def listCustomers(self) -> List[Customer]:
194
195     cursor = self.connection.cursor()
196
197     try:
198         cursor.execute("SELECT * FROM customer")
199         rows = cursor.fetchall()
200
201         customers = []
202         for row in rows:
203             customer = Customer(row[0], row[1], row[2], row[3], row[4])
204             customers.append(customer)
205
206         return customers
207     finally:
208         cursor.close()
209
210 2 usages
211 def findCarById(self, customerID: int) -> Customer:
212     IcarLeaseRepositoryImpl().findCarById()
```

The screenshot shows the same IDE with the 'icarleaseRepositoryImpl.py' file. The code defines two methods: 'findCustomerById' and 'createLease'. The 'findCustomerById' method takes a 'customerID' as an argument and returns a 'Customer' object. It uses a database cursor to execute a 'SELECT * FROM customer WHERE customerID=%s' query and fetches the first row. If a row is found, it returns a 'Customer' object; otherwise, it raises a 'CustomerNotFoundException'. The 'createLease' method takes 'customerID', 'carID', 'startDate', 'endDate', and 'lease_type' as arguments and returns a 'Lease' object. It uses a database cursor to execute an 'INSERT INTO lease' query with the provided values. After committing the transaction, it fetches the last inserted ID and returns a 'Lease' object with the appropriate values. The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces, and the Python version is 3.12.

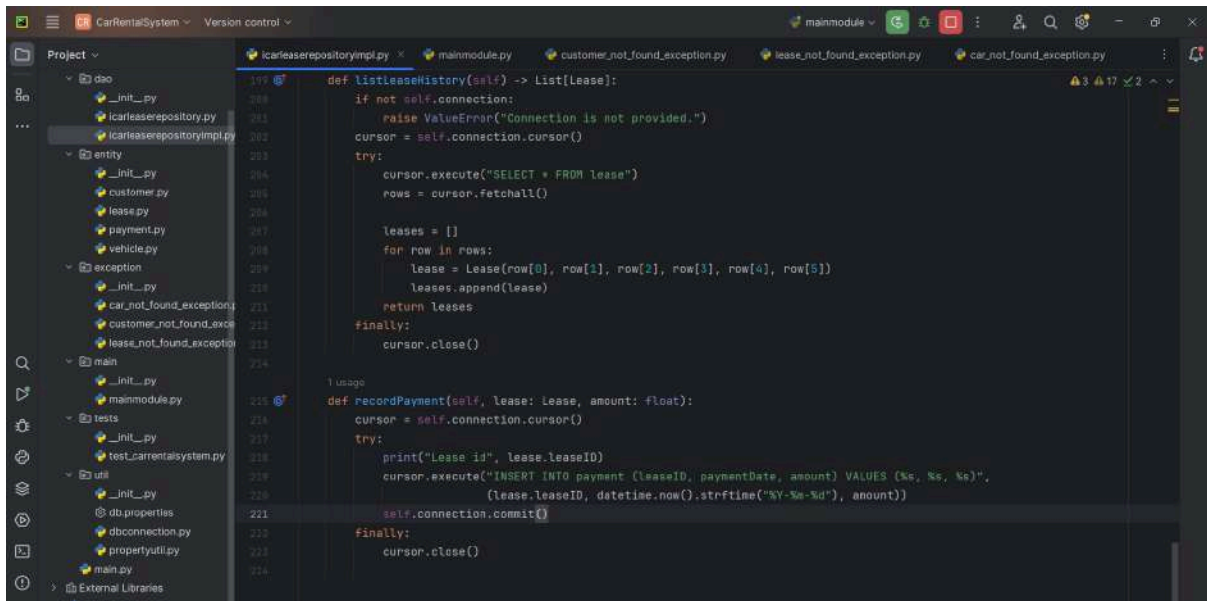
```
213 2 usages
214 def findCustomerById(self, customerID: int) -> Customer:
215
216     cursor = self.connection.cursor()
217
218     try:
219         cursor.execute("SELECT * FROM customer WHERE customerID=%s", (customerID,))
220         row = cursor.fetchone()
221
222         if row:
223             return Customer(row[0], row[1], row[2], row[3], row[4])
224         else:
225             raise CustomerNotFoundException(f"Customer with ID {customerID} not found.")
226     finally:
227         cursor.close()
228
229 2 usages
230 def createLease(self, customerID: int, carID: int, startDate, endDate, lease_type) -> Lease:
231
232     cursor = self.connection.cursor()
233
234     try:
235         cursor.execute("INSERT INTO lease (vehicleID, customerID, startDate, endDate, type) VALUES (%s, %s, %s, %s, %s)",
236                        (carID, customerID, startDate, endDate, lease_type))
237         self.connection.commit()
238         cursor.execute("SELECT LAST_INSERT_ID()")
239         leaseID = cursor.fetchone()[0]
240
241         return Lease(leaseID, carID, customerID, startDate, endDate)
```

The screenshot shows the IDE with the `ICarLeaseRepositoryImpl.py` file open. The `returnCar` method is being implemented. It takes `leaseID` as an argument and returns a `Lease` object. The method uses a database cursor to update the `endDate` of the lease to the current date. It also includes a `finally` block to close the cursor. The `dao` package structure is visible in the left sidebar, including `__init__.py`, `icarleaserepository.py`, `icarleaserepositoryimpl.py`, `entity`, `exception`, `main`, `tests`, and `util`.

```
195
196
197         return Lease(leaseID, carID, customerID, startDate, endDate,
198                     lease_type)
199
200     finally:
201         cursor.close()
202
203 1 usage
204 def returnCar(self, leaseID: int) -> Lease:
205
206     cursor = self.connection.cursor()
207
208     try:
209         cursor.execute("UPDATE lease SET endDate = %s WHERE leaseID = %s",
210                       (datetime.now().date(), leaseID))
211         self.connection.commit()
212
213         return self.findLeaseById(leaseID)
214     finally:
215         cursor.close()
216
217 5 usages
218 def findLeaseById(self, leaseID: int) -> Lease:
219     cursor = self.connection.cursor()
220
221     try:
222         cursor.execute("SELECT * FROM lease WHERE leaseID = %s", (leaseID,))
223         result = cursor.fetchone()
224
225         lease_id, car_id, customer_id, start_date, end_date, type = result
226         return Lease(lease_id, car_id, customer_id, start_date, end_date, type)
227     except:
228         raise LeaseNotFoundException(f"Lease with ID {leaseID} not found.")
229     finally:
230         cursor.close()
```

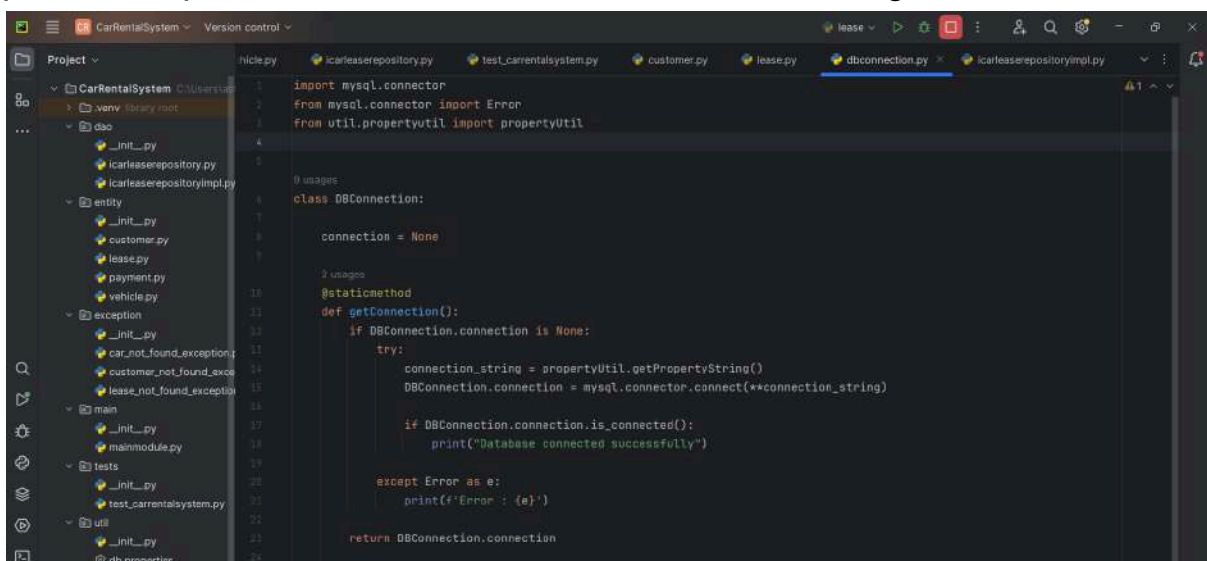
The screenshot shows the IDE with the `ICarLeaseRepositoryImpl.py` file open. The `findLeaseById` method is being implemented. It takes `leaseID` as an argument and returns a `Lease` object. The method uses a database cursor to fetch the lease details. It also includes a `finally` block to close the cursor. The `dao` package structure is visible in the left sidebar, including `__init__.py`, `icarleaserepository.py`, `icarleaserepositoryimpl.py`, `entity`, `exception`, `main`, `tests`, and `util`.

```
231
232
233     cursor.close()
234
235 5 usages
236 def findLeaseById(self, leaseID: int) -> Lease:
237     cursor = self.connection.cursor()
238
239     try:
240         cursor.execute("SELECT * FROM lease WHERE leaseID = %s", (leaseID,))
241         result = cursor.fetchone()
242
243         if result:
244             lease_id, car_id, customer_id, start_date, end_date, type = result
245             return Lease(lease_id, car_id, customer_id, start_date, end_date, type)
246         else:
247             raise LeaseNotFoundException(f"Lease with ID {leaseID} not found.")
248     finally:
249         cursor.close()
250
251 1 usage
252 def listActiveLeases(self) -> List[Lease]:
253     cursor = self.connection.cursor()
254
255     try:
256         cursor.execute("SELECT * FROM lease WHERE endDate >= %s", (datetime.now().date(),))
257         rows = cursor.fetchall()
258         leases = []
259
260         for row in rows:
261             lease_id, car_id, customer_id, start_date, end_date, type = row
262             lease = Lease(lease_id, car_id, customer_id, start_date, end_date, type)
263             leases.append(lease)
264
265     finally:
266         cursor.close()
267
268     return leases
```

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.



```

1 class PropertyUtil:
2     @staticmethod
3     def getPropertyString():
4         connection_string = {
5             'host': 'localhost',
6             'database': 'carrentalsystem',
7             'user': 'root',
8             'password': 'ABcd300#$',
9         }
10
11     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.
 - CustomerNotFoundException: throw this exception when user enters an invalid customer id which doesn't exist in db.

```

1 class CarNotFoundException(Exception):
2     def __init__(self, message):
3         super().__init__(message)

```

```

1 class CustomerNotFoundException(Exception):
2     def __init__(self, message):
3         super().__init__(message)

```

```

1 class LeaseNotFoundException(Exception):
2     def __init__(self, message):
3         super().__init__(message)

```

- main

```
1 from dao.icarleaseRepositoryImpl import ICarLeaseRepositoryImpl
2 from entity.vehicle import Vehicle
3 from entity.customer import Customer
4 from exception.car_not_found_exception import CarNotFoundException
5 from exception.customer_not_found_exception import CustomerNotFoundException
6 from exception.lease_not_found_exception import LeaseNotFoundException
7 from util.dbconnection import DBConnection
8
9
10 class MainModule:
11
12     def __init__(self):
13         self.car_repository = ICarLeaseRepositoryImpl(DBConnection.getConnection())
14
15     def display_menu(self):
16         print("\nCar Rental System Menu:")
17         print("1. Add a Car")
18         print("2. Remove a Car")
19         print("3. List Available Cars")
20         print("4. List Rented Cars")
21         print("5. Find Car by ID")
22         print("6. Add a Customer")
23         print("7. Remove a Customer")
24         print("8. List Customers")
25         print("9. Find Customer by ID")
26         print("10. Create Lease")
27         print("11. Return Car")
```

```
28
29 print("13. List Lease History")
30 print("14. Record Payment")
31 print("15. Exit")
32
33
34 def run(self):
35     while True:
36         self.display_menu()
37         choice = input("Enter your choices (1-15): ")
38
39         if choice == "15":
40             print("Exiting Car Rental System. Goodbye!")
41             break
42         elif choice == "1":
43             self.add_car()
44         elif choice == "2":
45             self.remove_car()
46         elif choice == "3":
47             self.list_available_cars()
48         elif choice == "4":
49             self.list_rented_cars()
50         elif choice == "5":
51             self.find_car_by_id()
52         elif choice == "6":
53             self.add_customer()
54         elif choice == "7":
55             self.remove_customer()
56         elif choice == "8":
57             self.list_customers()
```

```

85         elif choice == "8":
86             self.list_customers()
87         elif choice == "9":
88             self.find_customer_by_id()
89         elif choice == "10":
90             self.create_lease()
91         elif choice == "11":
92             self.return_car()
93         elif choice == "12":
94             self.list_active_leases()
95         elif choice == "13":
96             self.list_lease_history()
97         elif choice == "14":
98             self.record_payment()
99         else:
100             print("Invalid choice. Please enter a valid option.")
101
102 1 usage
103 def add_car(self):
104     print("Adding a new Car:")
105     make = input("Enter Make: ")
106     model = input("Enter Model: ")
107     year = input("Enter Year: ")
108     daily_rate = input("Enter Daily Rate: ")
109     status = input("Enter Status (available/notAvailable): ")
110     passenger_capacity = input("Enter Passenger Capacity: ")
111     engine_capacity = input("Enter Engine Capacity: ")
112
113     new_car = Vehicle(vehicle_id=None, make=make, model=model, year=year, daily_rate=daily_rate, status=status, passenger_capacity=passenger_capacity, engine_capacity=engine_capacity)

```

```

114     status = input("Enter Status (available/notAvailable): ")
115     passenger_capacity = input("Enter Passenger Capacity: ")
116     engine_capacity = input("Enter Engine Capacity: ")
117
118     new_car = Vehicle(vehicle_id=None, make=make, model=model, year=year, daily_rate=daily_rate, status=status, passenger_capacity=passenger_capacity, engine_capacity=engine_capacity)
119     self.car_repository.addCar(new_car)
120     print("Great! Car added successfully.")
121
122 1 usage
123 def remove_car(self):
124     print("Remove a Car:")
125     car_id = int(input("Enter Car ID to remove: "))
126
127     try:
128         self.car_repository.removeCar(car_id)
129         print("Car removed successfully.")
130     except CarNotFoundExcpetion as e:
131         print(f"Error: {e}")
132
133 1 usage
134 def list_available_cars(self):
135     print("List Available Cars:")
136     available_cars = self.car_repository.listAvailableCars()
137
138     if available_cars:
139         print("Available Cars:")
140         for car in available_cars:
141             print(f"{car.vehicleID}: {car.make} {car.model} - Status: {car.status}")
142     else:

```



```
183         print(f"{car.vehicleID}: {car.make} {car.model} - Status: {car.status}")
184     else:
185         print("No available cars.")
186
187 1 usage
188 def list_rented_cars(self):
189     print("List Rented Cars:")
190     rented_cars = self.car_repository.listRentedCars()
191
192     if rented_cars:
193         print("Rented Cars:")
194         for car in rented_cars:
195             print(f"{car.vehicleID}: {car.make} {car.model} - Status: {car.status}")
196     else:
197         print("No rented cars.")
198
199 1 usage
200 def find_car_by_id(self):
201     print("Find Car by ID:")
202     car_id = int(input("Enter Car ID to find: "))
203
204     try:
205         found_car = self.car_repository.findCarById(car_id)
206         print(f"Car Found: {found_car.make} {found_car.model} - Status: {found_car.status}")
207     except CarNotFoundException as e:
208         print(f"Error: {e}")
209
210 1 usage
211 def add_customer(self):
```

```
212 1 usage
213 def add_customer(self):
214     print("Add a Customer:")
215     first_name = input("Enter First Name: ")
216     last_name = input("Enter Last Name: ")
217     email = input("Enter Email: ")
218     phone_number = input("Enter Phone Number: ")
219
220     new_customer = Customer(customer_id=None, first_name, last_name, email, phone_number)
221     self.car_repository.addCustomer(new_customer)
222     print("Customer added successfully.")
223
224 1 usage
225 def remove_customer(self):
226     print("Remove a Customer:")
227     customer_id = int(input("Enter Customer ID to remove: "))
228
229     try:
230         self.car_repository.removeCustomer(customer_id)
231         print("Customer removed successfully.")
232     except CustomerNotFoundException as e:
233         print(f"Error: {e}")
234
235 1 usage
236 def list_customers(self):
237     print("List Customers:")
238     customers = self.car_repository.listCustomers()
239
240     if customers:
241         print("Customers:")
242         for customer in customers:
243             print(f"{customer.vehicleID}: {customer.make} {customer.model} - Status: {customer.status}")
244     else:
245         print("No customers found.")
246
247 1 usage
248 def list_customers():
```

The screenshot shows an IDE with a project named 'CarRentalSystem'. The left sidebar displays a file explorer with a tree structure: dao (dao.py, daoimpl.py), entity (entity.py, entityimpl.py), exception (exception.py, exceptionimpl.py), main (main.py, mainimpl.py), tests (tests.py, testsimpl.py), and util (util.py, utilimpl.py). The main editor window shows the implementation of the `find_customer_by_id` method in `mainmodule.py`. The code is as follows:

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

1 usage
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}")

1 usage
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

1 usage
MainModule - list_customers()
```

The screenshot shows the same IDE with the project 'CarRentalSystem'. The main editor window shows the implementation of the `return_car` method in `mainmodule.py`. The code is as follows:

```
raise e

1 usage
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}")

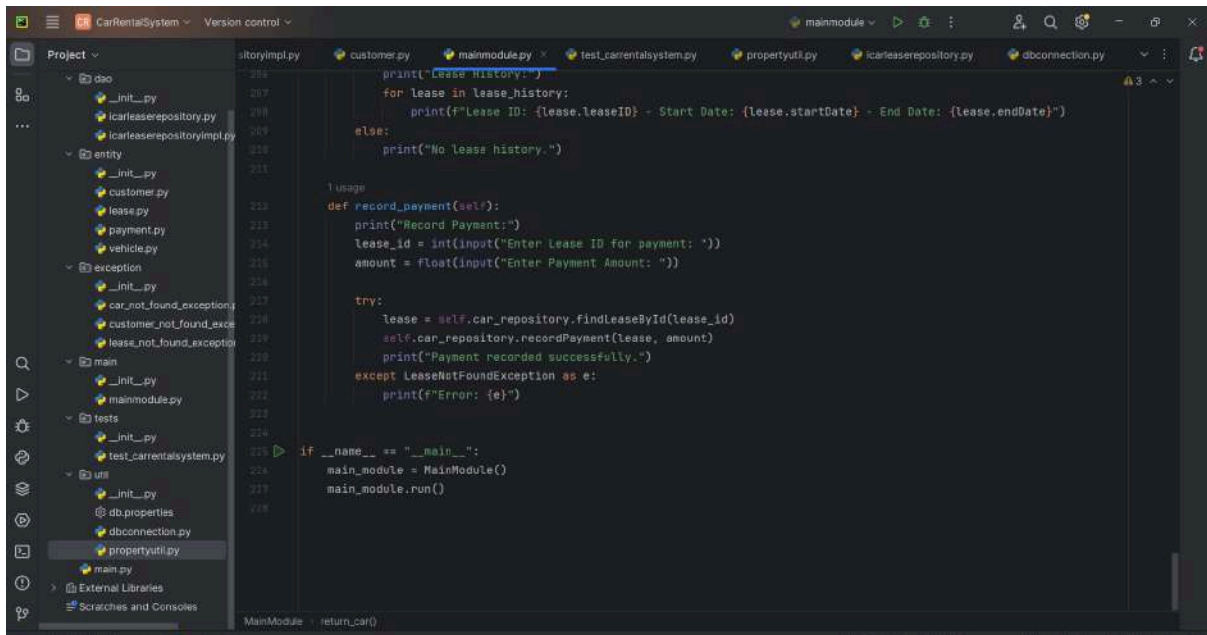
1 usage
def list_active_leases(self):
    print("List Active Leases:")
    active_leases = self.car_repository.listActiveLeases()

    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.")

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

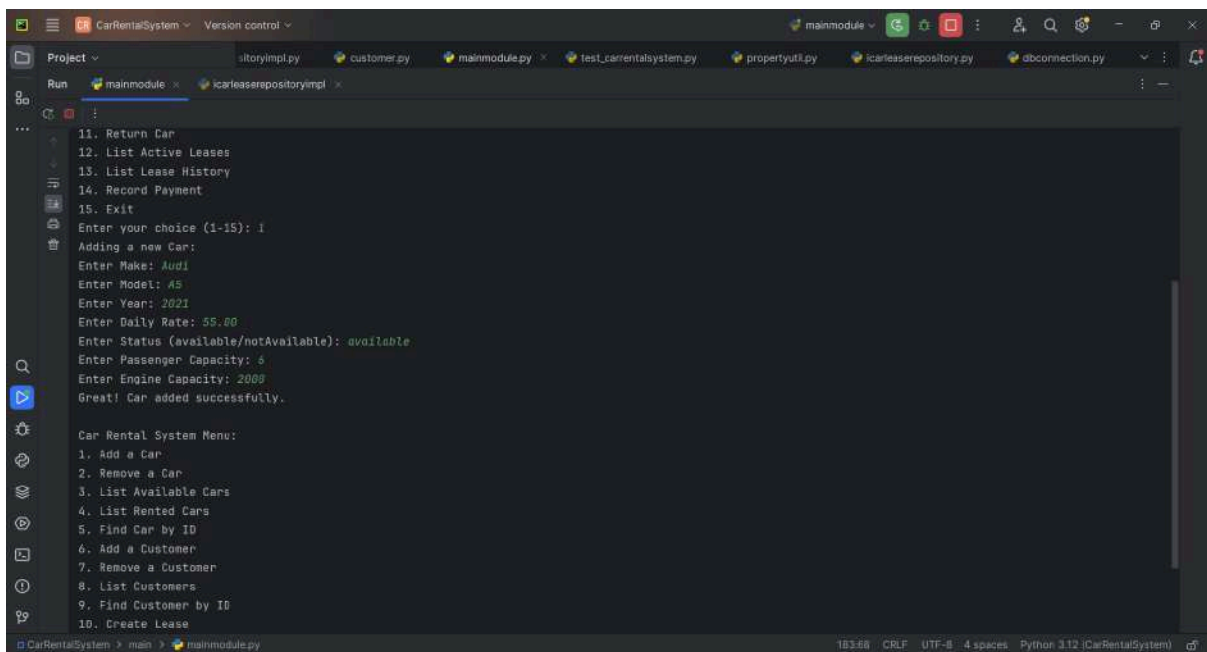
    if lease_history:
        print("Lease History:")
        for lease in lease_history:
            print(f"Lease ID: {lease.leaseID} - Start Date: {lease.startDate} - End Date: {lease.endDate}")

1 usage
MainModule - return_car()
```



```
216         print("Lease History..")
217         for lease in lease_history:
218             print(f"Lease ID: {lease.leaseID} - Start Date: {lease.startDate} - End Date: {lease.endDate}")
219     else:
220         print("No lease history.")
221
222 1 usage
223 def record_payment(self):
224     print("Record Payment:")
225     lease_id = int(input("Enter Lease ID for payment: "))
226     amount = float(input("Enter Payment Amount: "))
227
228     try:
229         lease = self.car_repository.findLeaseById(lease_id)
230         self.car_repository.recordPayment(lease, amount)
231         print("Payment recorded successfully.")
232     except LeaseNotFoundException as e:
233         print(f"Error: {e}")
234
235 if __name__ == "__main__":
236     main_module = MainModule()
237     main_module.run()
238
239 MainModule -> return_car()
```

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: $5.00
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
```

```
CarRentalSystem - Version control
mainmodule -
sitoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarlease repository.py dbconnection.py
Run mainmodule x icarlease repositoryimpl x
...
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): |
```

```
CarRentalSystem - Version control
mainmodule -
sitoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarlease repository.py dbconnection.py
Run mainmodule x icarlease repositoryimpl x
...
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
```

```
CarRentalSystem - Version control
mainmodule
sitoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarlease repository.py dbconnection.py
Run mainmodule x icarlease repositoryimpl x
...
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):
```

```
CarRentalSystem - Version control
mainmodule
sitoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarlease repository.py dbconnection.py
Run mainmodule x icarlease repositoryimpl x
...
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
```

```
CarRentalSystem - Version control
mainmodule x
sitoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarleasepository.py dbconnection.py
Run mainmodule xcarleasepositoryimpl x
...
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
```

```
CarRentalSystem - Version control
mainmodule x
sitoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarleasepository.py dbconnection.py
Run mainmodule xcarleasepositoryimpl x
...
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):
```



```
CarRentalSystem - Version control
mainmodule x itoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarleasepository.py dbconnection.py
Run mainmodule x icarleasepositoryimpl x
...
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
```

```
CarRentalSystem - Version control
mainmodule x itoryimpl.py customer.py mainmodule.py test_carrentalsystem.py propertyutil.py icarleasepository.py dbconnection.py
Run mainmodule x icarleasepositoryimpl x
...
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
```

```
CarRentalSystem ~ Version control ~ mainmodule ~ sitorimpl.py ~ customer.py ~ mainmodule.py ~ test_carrentalsystem.py ~ propertyutil.py ~ icarleasepository.py ~ dbconnection.py
Run mainmodule ~ icarleasepositoryimpl ~
...
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):
```

```
CarRentalSystem ~ Version control ~ mainmodule ~ sitorimpl.py ~ customer.py ~ mainmodule.py ~ test_carrentalsystem.py ~ propertyutil.py ~ icarleasepository.py ~ dbconnection.py
Run mainmodule ~ icarleasepositoryimpl ~
...
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
```



```
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:
Enter Lease ID for payment: 13
Enter Payment Amount: 1000
Lease id 13
Payment recorded 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): 15
Exiting Car Rental System. Goodbye!
Process finished with exit code 0
```

- 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 tests in test_carrentalsystem.py
11 usages
12 @pytest.fixture
13 def car_repository():
14     connection = DBConnection.getConnection()
15     return ICarLeaseRepositoryImpl(connection)
16 car_repository()

Run mainmodule Python tests in test_carrentalsystem.py
Test Results 11ms Tests passed: 6 of 6 tests - 11ms
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%]

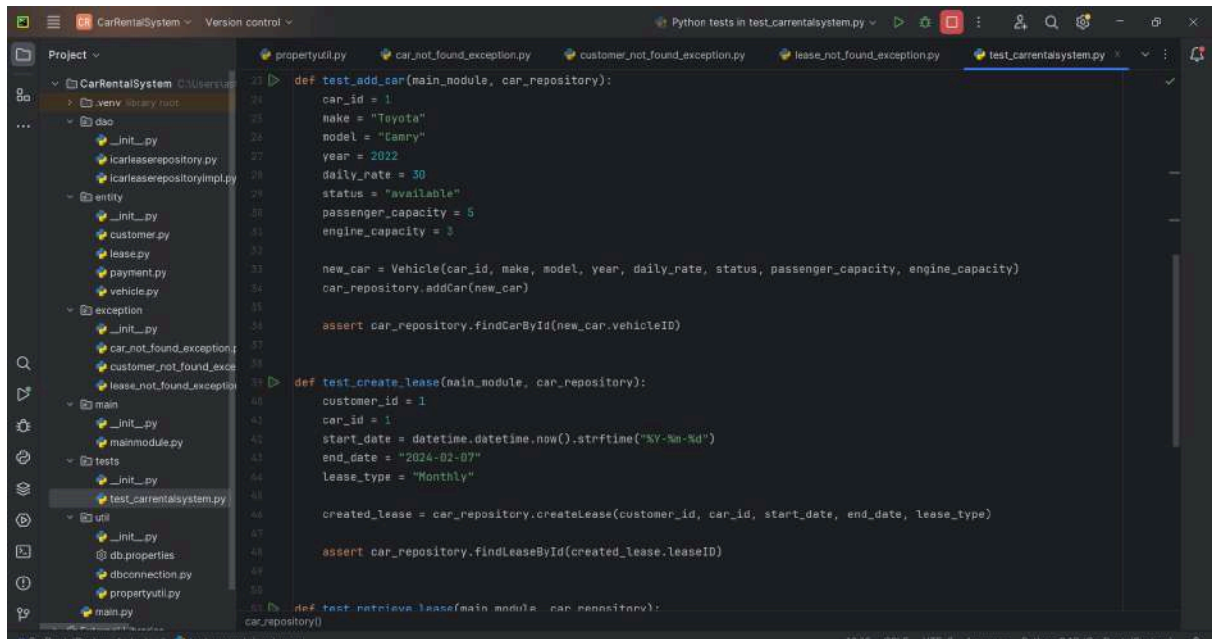
===== 6 passed in 0.18s =====

```

```

Python tests in test_carrentalsystem.py
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 CarNotFound
6 from exception.customer_not_found_exception import CustomerNotFound
7 from exception.lease_not_found_exception import LeaseNotFound
8 import datetime
9 from util.dbconnection import DBConnection
10
11 15 usages
12 @pytest.fixture
13 def car_repository():
14     connection = DBConnection.getConnection()
15     return ICarLeaseRepositoryImpl(connection)
16
17 6 usages
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()

```



```
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

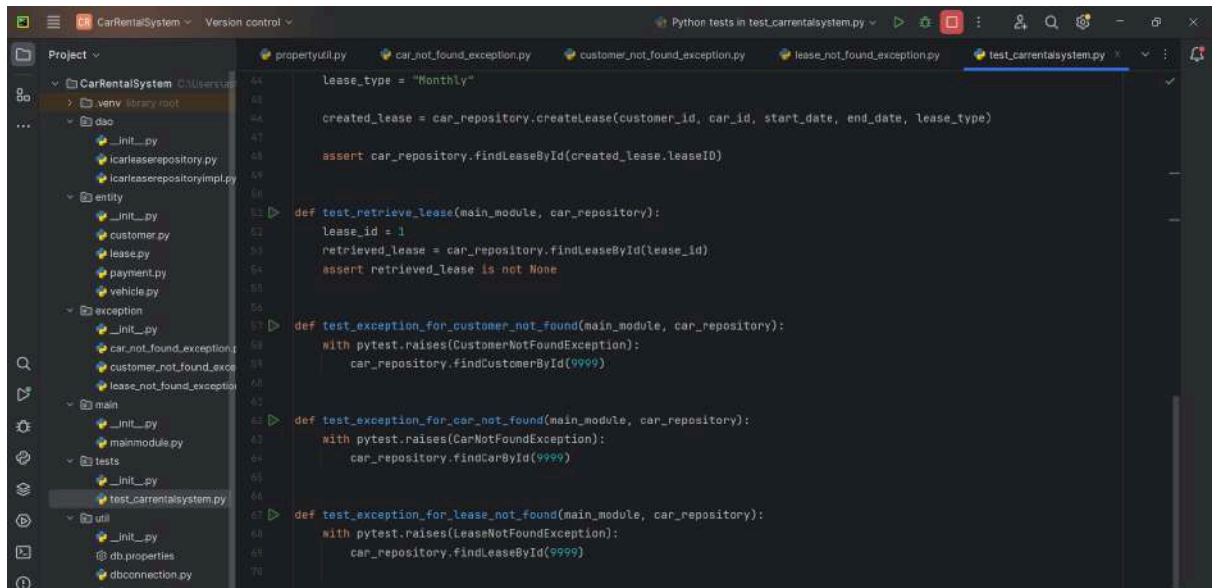
    new_car = Vehicle(car_id, make, model, year, daily_rate, status, passenger_capacity, engine_capacity)
    car_repository.addCar(new_car)

    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)
```

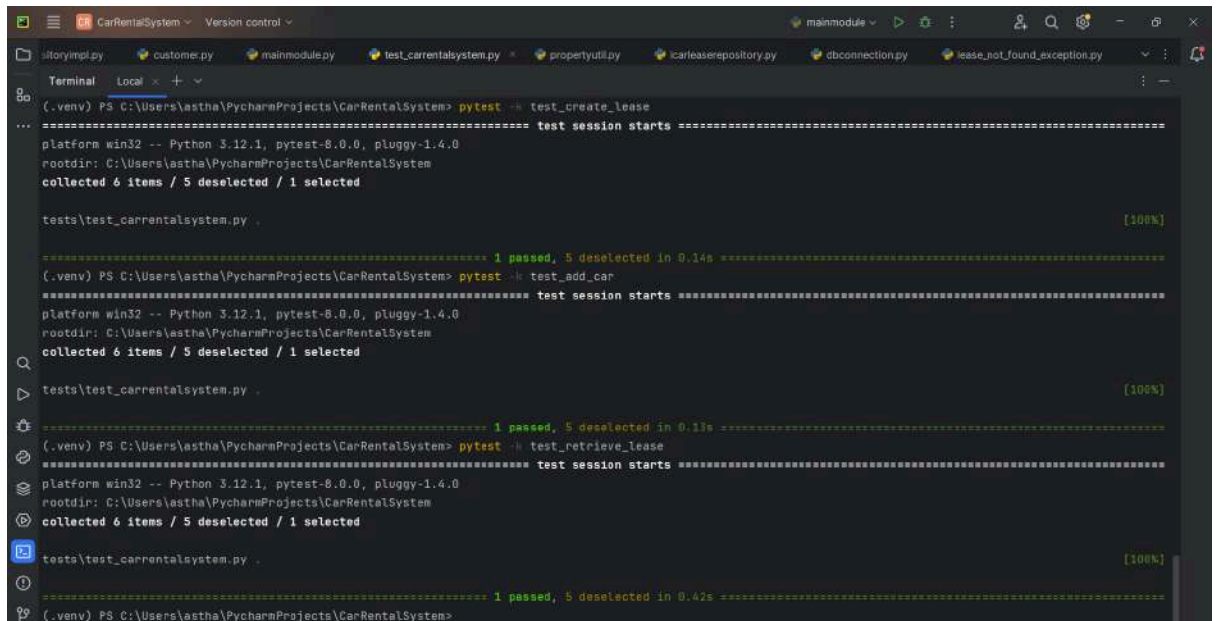


```
def test_retrieve_lease(main_module, car_repository):
    lease_id = 1
    retrieved_lease = car_repository.findLeaseById(lease_id)
    assert retrieved_lease is not None

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)
```



The screenshot shows a PyCharm IDE with a terminal window open. The terminal displays the output of three pytest test sessions. Each session shows the platform (win32), Python version (3.12.1), pytest version (8.0.0), and the number of collected items (6) and deselected items (5). The test results for each session are as follows:

Test Session	Test Name	Result	Duration
1	test_create_lease	1 passed, 5 deselected	0.14s
2	test_add_car	1 passed, 5 deselected	0.13s
3	test_retrieve_lease	1 passed, 5 deselected	0.42s

The terminal output is as follows:

```
(.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