Case study :Car Rental System

- Nila D

Synopsis:

Introduction

Project Overview

Functionality

Sql Part:

- **Entity Relationship Diagram**
- **Table Creation**
- **Values Inserted**

Python Part:

- **Structure**
- **Entity**
- **Dao**
- **Exception**
- **Util**
- **Main.Py**
- **Sample Outputs**

Unti Testing

Conclusion

1. Introduction

The **Car Rental System** is a console-based application designed to manage vehicle rentals efficiently. It allows customers to lease cars on a daily or monthly basis while handling car availability, customer records, lease details, and payments. The system is built using Python and MySQL, focusing on modular design, database interaction, and exception handling to simulate real-world car rental operations.

The **Car Rental System** is developed to streamline the management of a rental business through a command-line interface. The system is structured into multiple layers, including entity classes, data access objects (DAO), utility classes, exception handling, and a main module to interact with users.

Key components of the system include:

- **Customer Management**: Add, view, and remove customer details.
- **Car Management**: Maintain car records, update availability, and view available or rented cars.
- **Lease Management**: Create leases, return cars, and maintain lease history.
- **Payment Handling**: Record payments, retrieve customer payment history, and calculate revenue.

The system ensures organized database connectivity via utility classes and performs operations securely using custom exceptions and validation.

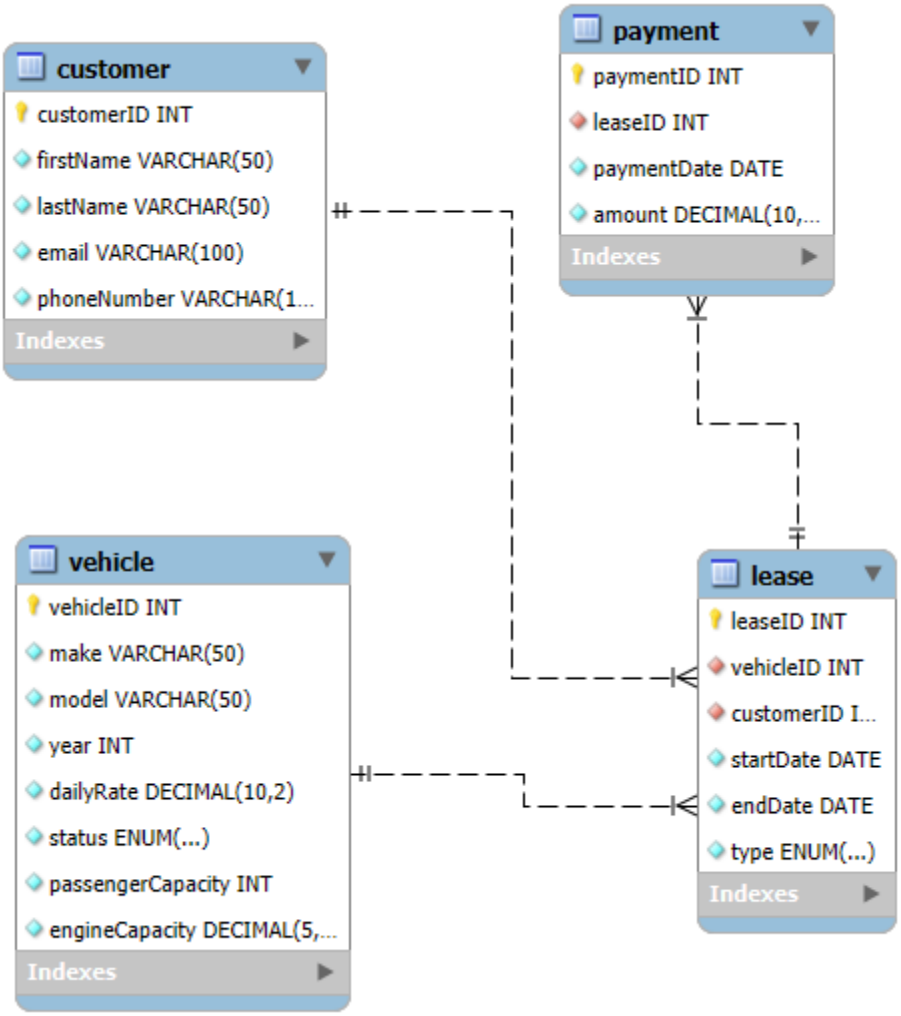
3. Functionality

The Car Rental System offers the following core functionalities:

1. **Customer Management**
    - **Add New Customers**: Register a new customer with details such as name, email, and phone number.
    - **Update Customer Information**: Modify customer details if required.
    - **Retrieve Customer Details**: View customer information by ID.
2. **Car Management**
    - **Add New Cars**: Add new vehicles to the fleet with attributes like make, model, year, and daily rate.
    - **Update Car Availability**: Set the car's status as available or not available.
    - **Retrieve Car Information**: View car details, including availability and capacity.
3. **Lease Management**
    - **Create Leases**: Customers can lease cars either daily or monthly by specifying start and end dates.
    - **Calculate Lease Cost**: Compute the cost based on the lease type (daily/monthly) and duration.
    - **Return Car**: Complete the lease and return the car, updating the system accordingly.
4. **Payment Handling**
    - **Record Payments**: Track payments for leases and store payment history.
    - **Retrieve Payment History**: View all payments made by a customer.
    - **Calculate Total Revenue**: Compute the total revenue generated from recorded payments.

SQL Part - Entity Relationship Diagram (ER Diagram)

An **Entity-Relationship (ER) Diagram** represents the data model and shows how entities like **Customer**, **Vehicle**, **Lease**, and **Payment** relate to each other in the database.



1. **Customer Entity**:
    - Attributes: customerID, firstName, lastName, email, phoneNumber
    - **Primary Key**: customerID
2. **Vehicle Entity**:
    - Attributes: vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity
    - **Primary Key**: vehicleID
3. **Lease Entity**:
    - Attributes: leaseID, vehicleID, customerID, startDate, endDate, type
    - **Primary Key**: leaseID
    - **Foreign Keys**: vehicleID (references Vehicle), customerID (references Customer)
4. **Payment Entity**:

- Attributes: paymentID, leaseID, paymentDate, amount
- **Primary Key**: paymentID
- **Foreign Key**: leaseID (references Lease)

Relationships:

- **Customer ↔ Lease**: A customer can have multiple leases. (1-to-many)
- **Vehicle ↔ Lease**: A vehicle can be rented in multiple leases, but each lease involves only one vehicle. (1-to-many)
- **Lease ↔ Payment**: Each lease can have multiple payments. (1-to-many)

Table Creation

Below are the SQL scripts used to create the database tables:

1. Vehicle Table

```
CREATE TABLE Vehicle (
    vehicleID INT PRIMARY KEY AUTO_INCREMENT,
    make VARCHAR(50) NOT NULL,
    model VARCHAR(50) NOT NULL,
    year INT NOT NULL,
    dailyRate DECIMAL(10,2) NOT NULL,
    status ENUM('available', 'notAvailable') NOT NULL,
    passengerCapacity INT NOT NULL,
    engineCapacity DECIMAL(5,2) NOT NULL
);
```

2. Customer Table

```
CREATE TABLE Customer (
    customerID INT PRIMARY KEY AUTO_INCREMENT,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phoneNumber VARCHAR(15) UNIQUE NOT NULL
```

);

3. Lease Table

```
CREATE TABLE Lease (
   leaseID INT PRIMARY KEY AUTO_INCREMENT,
   vehicleID INT NOT NULL,
   customerID INT NOT NULL,
   startDate DATE NOT NULL,
   endDate DATE NOT NULL,
   type ENUM('DailyLease', 'MonthlyLease') NOT NULL,
   FOREIGN KEY (vehicleID) REFERENCES Vehicle(vehicleID) ON DELETE CASCADE,
   FOREIGN KEY (customerID) REFERENCES Customer(customerID) ON DELETE CASCADE
);
```

4. Payment Table

```
CREATE TABLE Payment (
   paymentID INT PRIMARY KEY AUTO_INCREMENT,
   leaseID INT NOT NULL,
   paymentDate DATE NOT NULL,
   amount DECIMAL(10,2) NOT NULL,
   FOREIGN KEY (leaseID) REFERENCES Lease(leaseID) ON DELETE CASCADE
);
```

Values Inserted

Sample data is inserted into each table to demonstrate the functionality of the system.

1. Customer Data

```
INSERT INTO Customer (firstName, lastName, email, phoneNumber) VALUES
('Nila', 'Sharma', 'nila.sharma@example.com', '9876543210'),
('Nisha', 'Iyer', 'nisha.iyer@example.com', '9876543211'),
('Praveen', 'Rao', 'praveen.rao@example.com', '9876543212'),
('Uma', 'Krishnan', 'uma.krishnan@example.com', '9876543213'),
('Dhana', 'Raj', 'dhana.raj@example.com', '9876543214');
```

2. Vehicle Data

INSERT INTO Vehicle (make, model, year, dailyRate, status, passengerCapacity, engineCapacity) VALUES

('Maruti Suzuki', 'Swift', 2022, 1500.00, 'available', 5, 1.2),

('Hyundai', 'Creta', 2023, 2500.00, 'available', 5, 1.5),

('Honda', 'City', 2021, 2000.00, 'notAvailable', 5, 1.5),

('Tata', 'Nexon', 2023, 1800.00, 'available', 5, 1.2),

('Mahindra', 'Scorpio', 2022, 3000.00, 'notAvailable', 7, 2.2);

3. Lease Data

INSERT INTO Lease (vehicleID, customerID, startDate, endDate, type) VALUES

(1, 1, '2025-04-01', '2025-04-10', 'DailyLease'),

(2, 2, '2025-04-05', '2025-04-20', 'MonthlyLease'),

(3, 3, '2025-04-02', '2025-04-07', 'DailyLease'),

(4, 4, '2025-04-06', '2025-04-30', 'MonthlyLease'),

(5, 5, '2025-04-08', '2025-04-15', 'DailyLease');

4. Payment Data

INSERT INTO Payment (leaseID, paymentDate, amount) VALUES

(1, '2025-04-01', 15000.00),

(2, '2025-04-05', 50000.00),

(3, '2025-04-02', 10000.00),

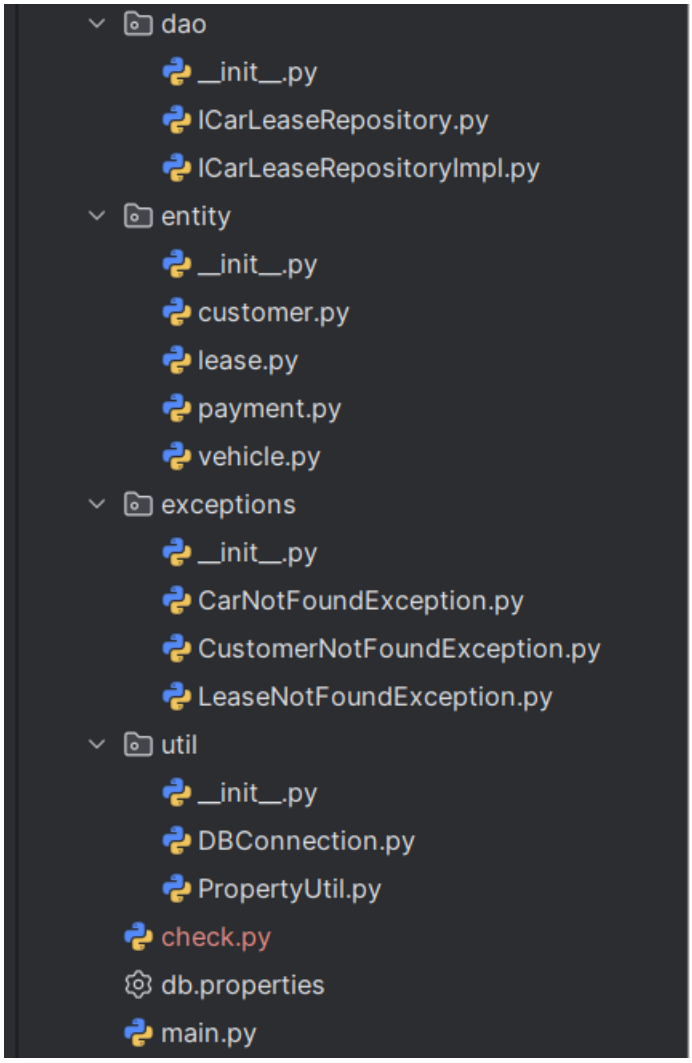
(4, '2025-04-06', 54000.00),

(5, '2025-04-08', 12000.00);

5. Python Part

Structure

The project is structured into multiple packages and modules to ensure clean, modular, and maintainable code.



Entity Classes

The **entity classes** represent the main data objects for the system. These classes contain only attributes and methods for data access (getters/setters), and no business logic.

1. Vehicle Class

- The `Vehicle` class represents a car or vehicle available for rent in the system. This class contains various attributes describing the car, such as its make, model, year, status, passenger capacity, and engine capacity. The class also provides getters and setters to access and modify these attributes.

```
class Vehicle:
    def __init__(self, make="", model="", year=0, dailyRate=0.0,
status="available", passengerCapacity=0, engineCapacity=0):
        # Initialize without vehicle_id since it's auto-incremented
in the DB
        self.__vehicleID = None  # Placeholder for vehicle_id, set
after DB insert
        self.__make = make
        self.__model = model
        self.__year = year
        self.__dailyRate = dailyRate
        self.__status = status
```

```
            self.__passengerCapacity = passengerCapacity
            self.__engineCapacity = engineCapacity

        # Getters
        def get_vehicleID(self):
            return self.__vehicleID
        def get_make(self):
            return self.__make
        def get_model(self):
            return self.__model
        def get_year(self):
            return self.__year
        def get_dailyRate(self):
            return self.__dailyRate
        def get_status(self):
            return self.__status
        def get_passengerCapacity(self):
            return self.__passengerCapacity
        def get_engineCapacity(self):
            return self.__engineCapacity
        # Setters
        def set_vehicleID(self, vehicleID):
            self.__vehicleID = vehicleID
        def set_make(self, make):
            self.__make = make
        def set_model(self, model):
            self.__model = model
        def set_year(self, year):
            self.__year = year
        def set_dailyRate(self, dailyRate):
            self.__dailyRate = dailyRate
        def set_status(self, status):
            self.__status = status
        def set_passengerCapacity(self, passengerCapacity):
            self.__passengerCapacity = passengerCapacity
        def set_engineCapacity(self, engineCapacity):
            self.__engineCapacity = engineCapacity
```

2. Customer Class

- The `Customer` class represents a customer in the car rental system. It contains information about the customer, such as their first name, last name, email, and phone number. The class also includes getters and setters to manage these properties.

```
class Customer:
    def __init__(self, customerID=0, firstName="", lastName="",
email="", phoneNumber=""):
        self.__customerID = customerID
        self.__firstName = firstName
        self.__lastName = lastName
        self.__email = email
        self.__phoneNumber = phoneNumber

    # Getters
    def get_customerID(self):
        return self.__customerID

    def get_firstName(self):
        return self.__firstName
```

```
    def get_lastName(self):
        return self.__lastName

    def get_email(self):
        return self.__email

    def get_phoneNumber(self):
        return self.__phoneNumber

    # Setters
    def set_customerID(self, customerID):
        self.__customerID = customerID

    def set_firstName(self, firstName):
        self.__firstName = firstName

    def set_lastName(self, lastName):
        self.__lastName = lastName

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

    def set_phoneNumber(self, phoneNumber):
        self.__phoneNumber = phoneNumber
```

3. Lease Class

- The `Lease` class represents a lease transaction for renting a vehicle. It contains details about the lease such as the lease ID, vehicle ID, customer ID, start date, end date, and lease type. It also provides getters and setters for accessing and modifying these values.

```
- class Lease:
    def __init__(self, leaseID=0, vehicleID=0, customerID=0,
startDate="", endDate="", leaseType="",active=True):
        self.__leaseID = leaseID
        self.__vehicleID = vehicleID
        self.__customerID = customerID
        self.__startDate = startDate
        self.__endDate = endDate
        self.__leaseType = leaseType  # "DailyLease" or
"MonthlyLease"
        self.__active = active  # ☑ Add this

        # Add getter/setter for active
        def is_active(self):
            return self.__active

        def set_active(self, active):
            self.__active = active

    # Getters
    def get_leaseID(self):
        return self.__leaseID

    def get_vehicleID(self):
        return self.__vehicleID

    def get_customerID(self):
        return self.__customerID
```

```
    def get_startDate(self):
        return self.__startDate

    def get_endDate(self):
        return self.__endDate

    def get_leaseType(self):
        return self.__leaseType

    # Setters
    def set_leaseID(self, leaseID):
        self.__leaseID = leaseID

    def set_vehicleID(self, vehicleID):
        self.__vehicleID = vehicleID

    def set_customerID(self, customerID):
        self.__customerID = customerID

    def set_startDate(self, startDate):
        self.__startDate = startDate

    def set_endDate(self, endDate):
        self.__endDate = endDate

    def set_leaseType(self, leaseType):
        self.__leaseType = leaseType
```

4. Payment Class

The Payment class represents a payment made for a specific lease. It contains details about the payment, including the payment ID, lease ID, payment amount, and the payment date. The class includes getters and setters to manage these attributes.

```
class Payment:
    def __init__(self, paymentID=0, leaseID=0, paymentDate="",
amount=0.0):
        self.__paymentID = paymentID
        self.__leaseID = leaseID
        self.__paymentDate = paymentDate
        self.__amount = amount

    # Getters
    def get_paymentID(self):
        return self.__paymentID

    def get_leaseID(self):
        return self.__leaseID

    def get_paymentDate(self):
        return self.__paymentDate

    def get_amount(self):
        return self.__amount

    # Setters
    def set_paymentID(self, paymentID):
```

```
        self.__paymentID = paymentID

    def set_leaseID(self, leaseID):
        self.__leaseID = leaseID

    def set_paymentDate(self, paymentDate):
        self.__paymentDate = paymentDate

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

DAO (Data Access Object) Layer

The DAO layer consists of interfaces and their corresponding implementations that handle database operations for each entity in the system. The goal of the DAO pattern is to abstract the database interaction and provide a cleaner way to access and manipulate data.

ICarLeaseRepository Interface

The ICarLeaseRepository interface defines the methods that the repository implementation must provide. It ensures that the data access layer is abstracted, so any changes to the database logic don't affect the rest of the application.

```
# dao/ICarLeaseRepository.py

from abc import ABC, abstractmethod
from entity.vehicle import Vehicle
from entity.customer import Customer
from entity.lease import Lease
from datetime import date

class ICarLeaseRepository(ABC):

    # 🚙 Car Management
    @abstractmethod
    def addCar(self, car: Vehicle) -> None:
        pass

    @abstractmethod
    def removeCar(self, carID: int) -> None:
        pass

    @abstractmethod
    def listAvailableCars(self) -> list:
        pass

    @abstractmethod
    def listRentedCars(self) -> list:
        pass

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

    # 👤 Customer Management
    @abstractmethod
```

```
    def addCustomer(self, customer: Customer) -> None:
        pass

    @abstractmethod
    def removeCustomer(self, customerID: int) -> None:
        pass

    @abstractmethod
    def listCustomers(self) -> list:
        pass

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

    # 📄 Lease Management
    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate:
date, endDate: date) -> Lease:
        pass

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

    @abstractmethod
    def listActiveLeases(self) -> list:
        pass

    @abstractmethod
    def listLeaseHistory(self) -> list:
        pass

    # 💳 Payment Handling
    @abstractmethod
    def recordPayment(self, lease: Lease, amount: float) -> None:
        pass
```

Abstract Methods:

- The ICarLeaseRepository interface declares abstract methods for all CRUD operations related to vehicles, customers, leases, and payments.
- Each entity has its own set of methods (e.g., addCar, removeCar, listAvailableCars for Vehicle).
- The interface is used by the repository implementation class to define the actual database interaction.

CarLeaseRepositoryImpl Class

- `CarLeaseRepositoryImpl` is the concrete implementation of the `ICarLeaseRepository` interface. It contains the actual SQL queries and the logic to interact with the MySQL database.

```
from dao.ICarLeaseRepository import ICarLeaseRepository
from entity.vehicle import Vehicle
from entity.customer import Customer
from entity.lease import Lease
from util.DBConnection import DBConnection  # ☑ Correct import
from datetime import date
import mysql.connector
```

```python
class CarLeaseRepositoryImpl(ICarLeaseRepository):

    def __init__(self):
        self.connection = DBConnection.getConnection()  # ☑ Correct
usage

    # 🚗 Car Management Methods
    def addCar(self, car):
        cursor = self.connection.cursor()

        # Removed vehicle_id from the INSERT query since it's auto-
generated by the database
        query = """INSERT INTO vehicle (make, model, year, dailyRate,
status, passengerCapacity, engineCapacity)
                   VALUES (%s, %s, %s, %s, %s, %s, %s)"""
        data = (
            car.get_make(), car.get_model(), car.get_year(),
            car.get_dailyRate(), car.get_status(),
            car.get_passengerCapacity(), car.get_engineCapacity()
        )
        cursor.execute(query, data)  # Execute the INSERT query
        car.set_vehicleID(cursor.lastrowid)  # Set the auto-generated
vehicle_id in the Vehicle object

        self.connection.commit()  # Commit the transaction to the
database

        print("☑ Car added successfully.")

    def removeCar(self, carID: int) -> None:
        cursor = self.connection.cursor()
        query = "DELETE FROM vehicle WHERE vehicleID = %s"
        cursor.execute(query, (carID,))
        self.connection.commit()
        print("☑ Car removed successfully.")

    def listAvailableCars(self) -> list:
        cursor = self.connection.cursor()
        query = "SELECT * FROM vehicle WHERE status = 'available'"  #
Use 'status' instead of 'available'
        cursor.execute(query)
        return cursor.fetchall()

    def listRentedCars(self) -> list:
        cursor = self.connection.cursor()
        query = "SELECT * FROM vehicle WHERE status = 'notAvailable'"
# Use 'status' instead of 'available = FALSE'
        cursor.execute(query)
        return cursor.fetchall()

    def findCarById(self, carID: int) -> Vehicle:
        cursor = self.connection.cursor()
        query = "SELECT * FROM vehicle WHERE vehicleID = %s"
        cursor.execute(query, (carID,))
        row = cursor.fetchone()
        if row:
            return Vehicle(*row)
        return None
```

```
    # 👤 Customer Management Methods
    def addCustomer(self, customer: Customer) -> None:
        cursor = self.connection.cursor()
        # Correct query (3 placeholders, no customer_id)
        query = "INSERT INTO customer (firstName, lastName, email,
phoneNumber) VALUES (%s, %s, %s, %s)"
        data = (
            customer.get_firstName(), customer.get_lastName(),
            customer.get_email(), customer.get_phoneNumber()
        )

        cursor.execute(query, data)
        self.connection.commit()
        print("✅ Customer added successfully.")

    def removeCustomer(self, customerID: int) -> None:
        cursor = self.connection.cursor()
        query = "DELETE FROM customer WHERE customerID = %s"
        cursor.execute(query, (customerID,))
        self.connection.commit()
        print("✅ Customer removed successfully.")

    def listCustomers(self) -> list:
        cursor = self.connection.cursor()
        query = "SELECT * FROM customer"
        cursor.execute(query)
        return cursor.fetchall()

    def findCustomerById(self, customerID: int) -> Customer:
        cursor = self.connection.cursor()
        query = "SELECT * FROM customer WHERE customerID = %s"
        cursor.execute(query, (customerID,))
        row = cursor.fetchone()
        if row:
            return Customer(*row)
        return None

    # 📜 Lease Management Methods
    def createLease(self, customerID: int, vehicleID: int, startDate:
date, endDate: date,
                    lease_type: str = "DailyLease") -> Lease:
        cursor = self.connection.cursor()

        query = """INSERT INTO lease (customerID, vehicleID,
startDate, endDate, type)
                   VALUES (%s, %s, %s, %s, %s)"""

        cursor.execute(query, (customerID, vehicleID, startDate,
endDate, lease_type))
        self.connection.commit()

        lease_id = cursor.lastrowid
        print("✅ Lease created successfully.")
        return Lease(lease_id, customerID, vehicleID, startDate,
endDate, lease_type)

    def returnCar(self, leaseID: int) -> Lease:
        cursor = self.connection.cursor()

        # 1. Mark lease as inactive
```

```
            query = "UPDATE lease SET active = FALSE WHERE leaseID = %s"
            cursor.execute(query, (leaseID,))

            # 2. Fetch lease info
            lease = self.findLeaseById(leaseID)

            # 3. Update vehicle status
            if lease:
                query = "UPDATE vehicle SET status = 'available' WHERE
vehicleID = %s"
                cursor.execute(query, (lease.get_vehicleID(),))

            self.connection.commit()
            print("✅ Lease returned and car status updated to
available.")
            return lease

        def listActiveLeases(self) -> list:
            cursor = self.connection.cursor()
            query = "SELECT * FROM lease WHERE active = TRUE"
            cursor.execute(query)
            return cursor.fetchall()

        def listLeaseHistory(self) -> list:
            cursor = self.connection.cursor()
            query = "SELECT * FROM lease WHERE active = FALSE"
            cursor.execute(query)
            return cursor.fetchall()

        def findLeaseById(self, leaseID: int) -> Lease:
            cursor = self.connection.cursor()
            query = "SELECT * FROM lease WHERE leaseId = %s"
            cursor.execute(query, (leaseID,))
            row = cursor.fetchone()
            if row:
                return Lease(*row)
            return None

        # 💰 Payment Handling
        def recordPayment(self, lease: Lease, amount: float) -> None:
            cursor = self.connection.cursor()
            query = "INSERT INTO payment (leaseID, amount, paymentDate)
VALUES (%s, %s, CURDATE())"
            cursor.execute(query, (lease.get_leaseID(), amount))
            self.connection.commit()
            print("✅ Payment recorded successfully.")
```

Explanation:

- **Car Management Methods:**
  - Methods like addCar, removeCar, listAvailableCars, and findCarById interact with the vehicle table to manage cars in the system.
- **Customer Management Methods:**
  - Methods like addCustomer, removeCustomer, listCustomers, and findCustomerById interact with the customer table.
- **Lease Management Methods:**

    - Methods like createLease, returnCar, listActiveLeases, listLeaseHistory, and findLeaseById interact with the lease table. The createLease method determines whether the lease is daily or monthly based on the duration.
- **Payment Handling:**
    - The recordPayment method inserts a payment record into the payment table for a specific lease.

Exception Handling

- In the Car Rental System project, custom exceptions are used to handle specific error scenarios related to the system's operations. These exceptions ensure that the application can handle errors gracefully, providing meaningful messages when operations fail.

1. CarNotFoundException

- This exception is raised when an operation cannot find a specific vehicle in the system. For instance, when a user attempts to retrieve or manipulate a vehicle that does not exist.

```
class CarNotFoundException(Exception):
    def __init__(self, message="Car with the given ID not found."):
        self.message = message
        super().__init__(self.message)
```

2. LeaseNotFoundException

- This exception is thrown when an operation cannot locate a specific lease. It is particularly useful when managing active or historical leases for cars and customers.

```
class LeaseNotFoundException(Exception):
    def __init__(self, message="Lease with the given ID not found."):
        self.message = message
        super().__init__(self.message)
```

3. CustomerNotFoundException

- This exception is raised when an operation cannot find a customer. It is useful when searching for customers in the system by their unique identifier.

```
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer with the given ID not
found."):
        self.message = message
        super().__init__(self.message)
```

Utility Classes

- The **util** package in the Car Rental System project contains two essential utility classes: `DBConnection` and `PropertyUtil`. These classes handle database connectivity and configuration management, ensuring the smooth functioning of the system.
- **1. DBConnection**

- `DBConnection` is responsible for establishing a connection to the MySQL database. It abstracts the details of setting up the database connection and provides a central place for accessing the connection throughout the application.

```
import mysql.connector
from mysql.connector import Error
from util.PropertyUtil import getPropertyString

class DBConnection:
    connection = None

    @staticmethod
    def getConnection():
        if DBConnection.connection is None:
            props = getPropertyString()
            print(" Loaded DB properties:", props)
            try:
                print(" Attempting to connect to DB...")
                DBConnection.connection = mysql.connector.connect(
                    host=props.get('host'),
                    port=props.get('port'),
                    database=props.get('database'),
                    user=props.get('user'),
                    password=props.get('password')
                )
                if DBConnection.connection:
                    print("Connected to database successfully!")
                else:
                    print(" Connection object is None.")
            except Error as e:
                print(f" Database connection failed: {e}")
        return DBConnection.connection
```

- **getConnection**: Establishes a connection to the MySQL database using `mysql.connector.connect`. It returns the database connection object.

- **Error Handling**: If the connection fails, an error message is printed, and `None` is returned.

2. PropertyUtil

PropertyUtil is designed to handle configuration settings by reading from a properties file, specifically bdbproperty. This utility simplifies the management of settings such as database configurations or other system-related parameters.

```
# util/PropertyUtil.py
def getPropertyString(filename='db.properties'):
    props = {}
    try:
        with open(filename, 'r') as f:
            for line in f:
                if line.strip() and not line.startswith('#'):
                    key, value = line.strip().split('=', 1)
                    props[key] = value
    except FileNotFoundError:
        print("❌ Properties file not found.")
    return props
```

- **get_property**: Reads from a property file (`bdbproperty.ini`) and retrieves the value associated with the specified property name.
- **Error Handling**: If the requested property is not found, an error message is printed, and `None` is returned.

Main.py :

```
# main/MainModule.py

from dao.ICarLeaseRepositoryImpl import CarLeaseRepositoryImpl
from entity.customer import Customer
from entity.vehicle import Vehicle
from datetime import datetime
import sys

def show_menu():
    print("\n===== Car Lease System Menu =====")
    print("1. Add Customer")
    print("2. Add Car")
    print("3. List Available Cars")
    print("4. Create Lease")
    print("5. Return Car")
    print("6. Record Payment")
    print("7. List Customers")
    print("8. List Rented Cars")
    print("9. Exit")

def main():
    repo = CarLeaseRepositoryImpl()

    while True:
        show_menu()
        choice = input("Enter your choice (1-9): ")

        if choice == '1':
            print("\n--- Add New Customer ---")
            first_name = input("First Name: ")
            last_name = input("Last Name: ")
            email = input("Email: ")
            phone = input("Phone Number: ")
            customer = Customer(firstName=first_name,
lastName=last_name, email=email, phoneNumber=phone)

            repo.addCustomer(customer)

        elif choice == '2':
            print("\n--- Add New Car ---")
            make = input("Make: ")
            model = input("Model: ")
            year = int(input("Year: "))
            daily_rate = float(input("Daily Rate: "))
            status = input("Status (available/notAvailable): ")
            passenger_capacity = int(input("Passenger Capacity: "))
            engine_capacity = float(input("Engine Capacity (cc): "))
            vehicle = Vehicle(make=make, model=model, year=year,
dailyRate=daily_rate,  # Use 'dailyRate' here
                              status=status,
passengerCapacity=passenger_capacity,
                              engineCapacity=engine_capacity)
```

```
            repo.addCar(vehicle)

        elif choice == '3':
            print("\n--- Available Cars ---")
            available_cars = repo.listAvailableCars()
            for car in available_cars:
                print(car)

        elif choice == '4':
            print("\n--- Create Lease ---")
            try:
                customer_id = int(input("Customer ID: "))
                vehicleID = int(input("Car ID: "))
                start_date = input("Start Date (YYYY-MM-DD): ")
                end_date = input("End Date (YYYY-MM-DD): ")
                lease = repo.createLease(customer_id, vehicleID,
start_date, end_date)
                print("Lease Created:", lease)
            except Exception as e:
                print(" Error:", e)

        elif choice == '5':
            print("\n--- Return Car ---")
            lease_id = int(input("Lease ID: "))
            try:
                returned = repo.returnCar(lease_id)
                print("Car Returned. Lease Info:", returned)
            except Exception as e:
                print("Error:", e)


        elif choice == '6':
            print("\n--- Record Payment ---")
            leaseId = int(input("Lease ID: "))
            amount = float(input("Payment Amount: "))
            try:
                lease = repo.findLeaseById(leaseId)  # Fetch the
Lease object by leaseId
                if lease:  # Check if the Lease object exists
                    repo.recordPayment(lease, amount)  # Pass the
Lease object to recordPayment
                    print("Payment Recorded.")
                else:
                    print("Lease not found.")
            except Exception as e:
                print("Error:", e)

        elif choice == '7':
            print("\n--- List of Customers ---")
            customers = repo.listCustomers()
            for customer in customers:
                print(customer)

        elif choice == '8':
            print("\n--- Rented Cars ---")
            rented = repo.listRentedCars()
            for car in rented:
                print(car)

        elif choice == '9':
```

```
                print(" Exiting application. Goodbye!")
                sys.exit(0)

            else:
                print("Invalid choice. Please select a number from 1 to
9.")

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

Output:

I have specified all the menu driven actions and how they have changed the database by using screenshots.

Task one : add customer

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 1

--- Add New Customer ---
First Name: aaron
Last Name: warner
Email: aaron123@gmail.com
Phone Number: 1234567890
✅ Customer added successfully.
```

The customer is added

customerID	firstName	lastName	email	phoneNumber
1	Nila	Sharma	nila.sharma@example.com	9876543210
2	Nisha	Iyer	nisha.iyer@example.com	9876543211
3	Praveen	Rao	praveen.rao@example.com	9876543212
4	Uma	Krishnan	uma.krishnan@example.com	9876543213
5	Dhana	Raj	dhana.raj@example.com	9876543214
6	aaron	warner	aaron123@gmail.com	1234567890
NULL	NULL	NULL	NULL	NULL

Task two: add car

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 2

--- Add New Car ---
Make: bmw
Model: bmw
Year: 2024
Daily Rate: 50000
Status (available/notAvailable): available
Passenger Capacity: 5
Engine Capacity (cc): 3
✅ Car added successfully.
```

We can check that by task 3 : list available car.

```
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 3

--- Available Cars ---
(1, 'Maruti Suzuki', 'Swift', 2022, Decimal('1500.00'), 'available', 5, Decimal('1.20'))
(2, 'Hyundai', 'Creta', 2023, Decimal('2500.00'), 'available', 5, Decimal('1.50'))
(4, 'Tata', 'Nexon', 2023, Decimal('1800.00'), 'available', 5, Decimal('1.20'))
(6, 'bmw', 'bmw', 2024, Decimal('50000.00'), 'available', 5, Decimal('3.00'))
```

Task four:create lease

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 4

--- Create Lease ---
Customer ID: 3
Car ID: 4
Start Date (YYYY-MM-DD): 2025-4-10
End Date (YYYY-MM-DD): 2025-04-15
✅ Lease created successfully.
```

Lease is created

	leaseID	vehicleID	customerID	startDate	endDate	type	active
▶	1	1	1	2025-04-01	2025-04-10	DailyLease	1
	2	2	2	2025-04-05	2025-04-20	MonthlyLease	1
	3	3	3	2025-04-02	2025-04-07	DailyLease	1
	4	4	4	2025-04-06	2025-04-30	MonthlyLease	1
	5	5	5	2025-04-08	2025-04-15	DailyLease	1
	6	4	3	2025-04-10	2025-04-15	DailyLease	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Task 5 : return car

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 5

--- Return Car ---
Lease ID: 5
✅ Lease returned and car status updated to available.
```

We can see the change in database

leaseID	vehicleID	customerID	startDate	endDate	type	active
1	1	1	2025-04-01	2025-04-10	DailyLease	1
2	2	2	2025-04-05	2025-04-20	MonthlyLease	1
3	3	3	2025-04-02	2025-04-07	DailyLease	1
4	4	4	2025-04-06	2025-04-30	MonthlyLease	1
5	5	5	2025-04-08	2025-04-15	DailyLease	0
6	4	3	2025-04-10	2025-04-15	DailyLease	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Task 6 : record payment

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 6

--- Record Payment ---
Lease ID: 6
Payment Amount: 9000
✅ Payment recorded successfully.
Payment Recorded.
```

paymentID	leaseID	paymentDate	amount
1	1	2025-04-01	15000.00
2	2	2025-04-05	50000.00
3	3	2025-04-02	10000.00
4	4	2025-04-06	54000.00
5	5	2025-04-08	12000.00
6	6	2025-04-10	9000.00
NULL	NULL	NULL	NULL

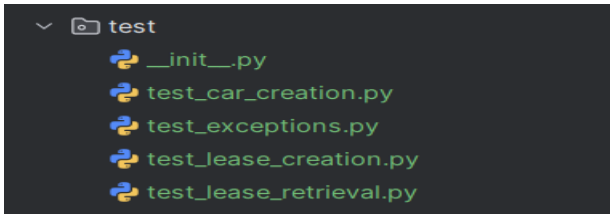
Tasl 7 :

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 7

--- List of Customers ---
(1, 'Nila', 'Sharma', 'nila.sharma@example.com', '9876543210')
(2, 'Nisha', 'Iyer', 'nisha.iyer@example.com', '9876543211')
(3, 'Praveen', 'Rao', 'praveen.rao@example.com', '9876543212')
(4, 'Uma', 'Krishnan', 'uma.krishnan@example.com', '9876543213')
(5, 'Dhana', 'Raj', 'dhana.raj@example.com', '9876543214')
(6, 'aaron', 'warner', 'aaron123@gmail.com', '1234567890')
```

Task 8:

```
===== Car Lease System Menu =====
1. Add Customer
2. Add Car
3. List Available Cars
4. Create Lease
5. Return Car
6. Record Payment
7. List Customers
8. List Rented Cars
9. Exit
Enter your choice (1-9): 8

--- Rented Cars ---
(3, 'Honda', 'City', 2021, Decimal('2000.00'), 'notAvailable', 5, Decimal('1.50'))
```

Unti testing:



unit testing is done for the car rental system. Specified in the code.