**SOURCE CODE REFERENCE**

**Project Title:** Car Rental System

**Document Owner:** Sujana D

**Date:** 26-06-2025

**TABLE OF CONTENTS**

1. **util**  
   1.1 DBPropertyUtil.py  
   1.2 DBConnUtil.py
2. **db.properties**
3. **entity**  
   3.1 customer.py  
   3.2 vehicle.py  
   3.3 lease.py  
   3.4 payment.py
4. **exception**  
   4.1 custom\_exceptions.py
5. **dao**  
   5.1 icar\_lease\_repository.py  
   5.2 icar\_lease\_repository\_impl.py
6. **tests**  
   6.1 test\_customer.py  
   6.2 test\_vehicle.py  
   6.3 test\_lease.py  
   6.4 test\_payment.py
7. **app**  
   7.1 main.py
8. **reporting**  
   8.1 generate\_test\_report.py

**1.** **util**

**1.1 DBPropertyUtil.py**

Purpose:

Reads database connection settings from db.properties and returns them as a dictionary.

**Code:**

**def get\_property\_string(file\_path):**

**props = {}**

**with open(file\_path, 'r') as file:**

**for line in file:**

**if '=' in line:**

**key, value = line.strip().split('=')**

**props[key] = value**

**return props**

**1.2 DBConnUtil.py**

Purpose:

Uses the properties from DBPropertyUtil to establish and return a MySQL database connection.

**Code:**

**import mysql.connector**

**from util.db\_property\_util import get\_property\_string**

**def get\_connection():**

**props = get\_property\_string('db.properties')**

**conn = mysql.connector.connect(**

**host=props['host'],**

**port=int(props['port']),**

**user=props['username'],**

**password=props['password'],**

**database=props['dbname']**

**)**

**return conn**

1. **db.properties**

**Purpose:**Holds key–value pairs for database credentials.

**Code:**

**host=localhost**

**port=3306**

**username=root**

**password=xxxxxxxx**

**dbname=CarRentalDB**

1. **entity  
   3.1 customer.py**

**Purpose:**Defines the Customer data model with no business logic.

**Code:**

**class Customer:**

**def \_\_init\_\_(self, customerID, firstName, lastName, email, phoneNumber):**

**self.customerID = customerID**

**self.firstName = firstName**

**self.lastName = lastName**

**self.email = email**

**self.phoneNumber = phoneNumber**

**def \_\_str\_\_(self):**

**return f"{self.customerID}: {self.firstName} {self.lastName} - {self.email} ({self.phoneNumber})"**

**3.2 vehicle.py**

**Purpose:**Defines the Vehicle data model.

**Code:**

**Class Vehicle:**

**def \_\_init\_\_(self, vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity):**

**self.vehicleID = vehicleID**

**self.make = make**

**self.model = model**

**self.year = year**

**self.dailyRate = dailyRate**

**self.status = status**

**self.passengerCapacity = passengerCapacity**

**self.engineCapacity = engineCapacity**

**def \_\_str\_\_(self):**

**return f"{self.vehicleID}: {self.make} {self.model} - ₹{self.dailyRate}/day - {self.status}"**

**3.3 lease.py**

**Purpose:**Defines the Lease data model.

**Code:**

**class Lease:**

**def \_\_init\_\_(self, leaseID, vehicleID, customerID, startDate, endDate, leaseType):**

**self.leaseID = leaseID**

**self.vehicleID = vehicleID**

**self.customerID = customerID**

**self.startDate = startDate**

**self.endDate = endDate**

**self.leaseType = leaseType**

**def \_\_str\_\_(self):**

**return f"Lease {self.leaseID} (Customer {self.customerID} - Vehicle {self.vehicleID})"**

**3.4 payment.py**

**Purpose:**Defines the Payment data model**.**

**Code:**

**class Payment:**

**def \_\_init\_\_(self, paymentID, leaseID, paymentDate, amount):**

**self.paymentID = paymentID**

**self.leaseID = leaseID**

**self.paymentDate = paymentDate**

**self.amount = amount**

**def \_\_str\_\_(self):**

**return f"Payment {self.paymentID} - ₹{self.amount} on {self.paymentDate}"**

1. **exception  
   4.1 custom\_exceptions.py**

**Purpose:**Declares domain-specific exceptions for “not found” conditions.

**Code:**

**class CustomerNotFoundException(Exception):**

**def \_\_init\_\_(self, customerID):**

**super().\_\_init\_\_(f"Customer with ID {customerID not found.")**

**class VehicleNotFoundException(Exception):**

**def \_\_init\_\_(self, vehicleID):**

**super().\_\_init\_\_(f"Vehicle with ID {vehicleID} not found.")**

**class LeaseNotFoundException(Exception):**

**def \_\_init\_\_(self, leaseID):**

**super().\_\_init\_\_(f"Lease with ID {leaseID} not found.")**

1. **dao  
   5.1 icar\_lease\_repository.py**

**Purpose:**Abstract interface defining CRUD and business methods for customers, vehicles, leases, and payments.

**Code:**

**from abc import ABC, abstractmethod**

**class ICarLeaseRepository(ABC):**

**# Customer Management**

**@abstractmethod**

**def addCustomer(self, customer): pass**

**@abstractmethod**

**def removeCustomer(self, customerID): pass**

**@abstractmethod**

**def listCustomers(self): pass**

**@abstractmethod**

**def findCustomerById(self, customerID): pass**

**# Car Management**

**@abstractmethod**

**def addCar(self, car): pass**

**@abstractmethod**

**def removeCar(self, carID): pass**

**@abstractmethod**

**def listAvailableCars(self): pass**

**@abstractmethod**

**def listRentedCars(self): pass**

**@abstractmethod**

**def findCarById(self, carID): pass**

**# Lease Management**

**@abstractmethod**

**def createLease(self, customerID, carID, startDate, endDate): pass**

**@abstractmethod**

**def returnCar(self, leaseID): pass**

**@abstractmethod**

**def listActiveLeases(self): pass**

**@abstractmethod**

**def listLeaseHistory(self): pass**

**# Payment Handling**

**@abstractmethod**

**def recordPayment(self, lease, amount): pass**

**5.2 icar\_lease\_repository\_impl.py**

**Purpose:**Concrete DAO implementation using MySQL. Raises custom exceptions when records are missing.

**Code:**

**from dao.icar\_lease\_repository import ICarLeaseRepository**

**from util.db\_conn\_util import get\_connection**

**from entity.customer import Customer**

**from entity.vehicle import Vehicle**

**from entity.lease import Lease**

**from entity.payment import Payment**

**from exception.custom\_exceptions import CustomerNotFoundException, VehicleNotFoundException, LeaseNotFoundException**

**class ICarLeaseRepositoryImpl(ICarLeaseRepository):**

**# -------------------------------**

**# CUSTOMER OPERATIONS**

**# -------------------------------**

**def addCustomer(self, customer):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**query = """**

**INSERT INTO Customer (firstName, lastName, email, phoneNumber)**

**VALUES (%s, %s, %s, %s)**

**"""**

**data = (customer.firstName, customer.lastName, customer.email, customer.phoneNumber)**

**cursor.execute(query, data)**

**conn.commit()**

**print("Customer added successfully!")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def removeCustomer(self, customerID):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("SELECT COUNT(\*) FROM Lease WHERE customerID = %s", (customerID,))**

**lease\_count = cursor.fetchone()[0]**

**if lease\_count > 0:**

**print(f"Cannot delete: Customer ID {customerID} has active or historical leases.")**

**return**

**cursor.execute("DELETE FROM Customer WHERE customerID = %s", (customerID,))**

**conn.commit()**

**if cursor.rowcount == 0:**

**raise CustomerNotFoundException(customerID)  # <-- Let this propagate!**

**else:**

**print(f"Customer ID {customerID} removed successfully.")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def listCustomers(self):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("SELECT \* FROM Customer")**

**customers = cursor.fetchall()**

**return customers**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def findCustomerById(self, customerID):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("SELECT \* FROM Customer WHERE customerID = %s", (customerID,))**

**customer = cursor.fetchone()**

**return customer**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**# -------------------------------**

**# VEHICLE OPERATIONS**

**# -------------------------------**

**def addCar(self, car):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**query = """**

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

**VALUES (%s, %s, %s, %s, %s, %s, %s)**

**"""**

**data = (car.make, car.model, car.year, car.dailyRate, car.status, car.passengerCapacity, car.engineCapacity)**

**cursor.execute(query, data)**

**conn.commit()**

**print("Vehicle added successfully!")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def removeCar(self, carID):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("DELETE FROM Vehicle WHERE vehicleID = %s", (carID,))**

**conn.commit()**

**if cursor.rowcount == 0:**

**raise VehicleNotFoundException(carID)  # Also propagate exception for vehicles!**

**else:**

**print(f"Car ID {carID} removed successfully.")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def listAvailableCars(self):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("SELECT \* FROM Vehicle WHERE status = 'available'")**

**result = cursor.fetchall()**

**return result**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def listRentedCars(self):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("""**

**SELECT V.\* FROM Vehicle V**

**JOIN Lease L ON V.vehicleID = L.vehicleID**

**WHERE CURDATE() BETWEEN L.startDate AND L.endDate**

**""")**

**result = cursor.fetchall()**

**return result**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def findCarById(self, carID):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("SELECT \* FROM Vehicle WHERE vehicleID = %s", (carID,))**

**car = cursor.fetchone()**

**if not car:**

**raise VehicleNotFoundException(carID)**

**return car**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**# -------------------------------**

**# LEASE OPERATIONS**

**# -------------------------------**

**def createLease(self, customerID, carID, startDate, endDate):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**query = """**

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

**VALUES (%s, %s, %s, %s, %s)**

**"""**

**lease\_type = 'Daily'**

**data = (carID, customerID, startDate, endDate, lease\_type)**

**cursor.execute(query, data)**

**conn.commit()**

**cursor.execute("UPDATE Vehicle SET status = 'notAvailable' WHERE vehicleID = %s", (carID,))**

**conn.commit()**

**print("Lease created successfully.")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def returnCar(self, leaseID):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**cursor.execute("SELECT vehicleID FROM Lease WHERE leaseID = %s", (leaseID,))**

**result = cursor.fetchone()**

**if result is None:**

**raise LeaseNotFoundException(leaseID)**

**vehicleID = result[0]**

**cursor.execute("UPDATE Lease SET endDate = CURDATE() WHERE leaseID = %s", (leaseID,))**

**cursor.execute("UPDATE Vehicle SET status = 'available' WHERE vehicleID = %s", (vehicleID,))**

**conn.commit()**

**print("Car returned successfully.")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def listActiveLeases(self):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**query = "SELECT \* FROM Lease WHERE CURDATE() BETWEEN startDate AND endDate"**

**cursor.execute(query)**

**result = cursor.fetchall()**

**return result**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**def listLeaseHistory(self):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**query = "SELECT \* FROM Lease WHERE endDate < CURDATE()"**

**cursor.execute(query)**

**result = cursor.fetchall()**

**return result**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

**# -------------------------------**

**# PAYMENT OPERATIONS**

**# -------------------------------**

**def recordPayment(self, lease, amount):**

**conn = get\_connection()**

**cursor = conn.cursor()**

**try:**

**query = """**

**INSERT INTO Payment (leaseID, paymentDate, amount)**

**VALUES (%s, CURDATE(), %s)**

**"""**

**cursor.execute(query, (lease, amount))**

**conn.commit()**

**print("Payment recorded successfully.")**

**finally:**

**if conn.is\_connected():**

**cursor.close()**

**conn.close()**

1. **tests**

**6.1 test\_customer.py**

**Purpose:**Validates customer CRUD and exception cases.

**Code:**

**import unittest**

**from entity.customer import Customer**

**from dao.icar\_lease\_repository\_impl import ICarLeaseRepositoryImpl**

**from exception.custom\_exceptions import CustomerNotFoundException**

**class TestCustomerOperations(unittest.TestCase):**

**@classmethod**

**def setUpClass(cls):**

**cls.repo = ICarLeaseRepositoryImpl()**

**cls.test\_customer = Customer(0, "Temp", "Test", "temp.test@example.com", "8888888888")**

**def test\_add\_and\_remove\_customer(self):**

**# Add**

**self.repo.addCustomer(self.test\_customer)**

**customers = self.repo.listCustomers()**

**new\_customer = [c for c in customers if c[2] == "Test"]**

**self.assertTrue(new\_customer)**

**new\_customer\_id = new\_customer[0][0]**

**# Remove**

**self.repo.removeCustomer(new\_customer\_id)**

**customers\_after = self.repo.listCustomers()**

**self.assertFalse(any(c[0] == new\_customer\_id for c in customers\_after))**

**def test\_remove\_customer\_not\_found(self):**

**with self.assertRaises(CustomerNotFoundException):**

**self.repo.removeCustomer(999999)  # unlikely to exist**

**def test\_find\_customer\_not\_found(self):**

**customer = self.repo.findCustomerById(999999)**

**self.assertIsNone(customer)  # or assertRaises if you throw exception**

**if \_\_name\_\_ == "\_\_main\_\_":**

**unittest.main()**

* 1. **test\_vehicle.py**

**Purpose:**Validates vehicle CRUD and exception cases.

**Code:**

**import unittest**

**from dao.icar\_lease\_repository\_impl import ICarLeaseRepositoryImpl**

**from exception.custom\_exceptions import VehicleNotFoundException**

**class TestVehicleOperations(unittest.TestCase):**

**@classmethod**

**def setUpClass(cls):**

**cls.repo = ICarLeaseRepositoryImpl()**

**def test\_remove\_vehicle\_not\_found(self):**

**with self.assertRaises(VehicleNotFoundException):**

**self.repo.removeCar(999999)**

**def test\_find\_vehicle\_not\_found(self):**

**with self.assertRaises(VehicleNotFoundException):**

**self.repo.findCarById(999999)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**unittest.main()**

**6.3 test\_lease.py**

**Purpose:**Validates lease creation, listing, return, and exception.

**Code:**

**import unittest**

**from dao.icar\_lease\_repository\_impl import ICarLeaseRepositoryImpl**

**from exception.custom\_exceptions import LeaseNotFoundException**

**class TestLeaseOperations(unittest.TestCase):**

**@classmethod**

**def setUpClass(cls):**

**cls.repo = ICarLeaseRepositoryImpl()**

**# can use valid customer and car IDs from your DB for creating test lease**

**cls.valid\_customer\_id = 1**

**cls.valid\_vehicle\_id = 2**

**def test\_create\_lease\_and\_list(self):**

**# Adjust dates as needed**

**start\_date = "2025-07-01"**

**end\_date = "2025-07-03"**

**self.repo.createLease(self.valid\_customer\_id, self.valid\_vehicle\_id, start\_date, end\_date)**

**active\_leases = self.repo.listActiveLeases()**

**# At least one active lease exists**

**self.assertTrue(isinstance(active\_leases, list))**

**def test\_list\_active\_leases(self):**

**result = self.repo.listActiveLeases()**

**self.assertIsInstance(result, list)**

**def test\_list\_lease\_history(self):**

**result = self.repo.listLeaseHistory()**

**self.assertIsInstance(result, list)**

**6.4 test\_payment.py**

**Purpose:**Validates payment recording.

**Code:**

**import unittest**

**from dao.icar\_lease\_repository\_impl import ICarLeaseRepositoryImpl**

**class TestPaymentOperations(unittest.TestCase):**

**@classmethod**

**def setUpClass(cls):**

**cls.repo = ICarLeaseRepositoryImpl()**

**def test\_record\_payment(self):**

**lease\_id = 1  # must exist in DB**

**self.repo.recordPayment(lease\_id, 500.00)**

**# Can't assert directly unless we fetch — placeholder success test**

**self.assertTrue(True)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**unittest.main()**

1. **app  
   7.1 main.py**

**Purpose:**Implements the menu-driven CLI, routes user choices to DAO methods, and handles exceptions.

**Code:**

**import sys**

**import os**

**sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))**

**from dao.icar\_lease\_repository\_impl import ICarLeaseRepositoryImpl**

**from entity.customer import Customer**

**from entity.vehicle import Vehicle**

**from exception.custom\_exceptions import CustomerNotFoundException, VehicleNotFoundException, LeaseNotFoundException**

**def main():**

**repo = ICarLeaseRepositoryImpl()**

**while True:**

**print("\n====== CAR RENTAL SYSTEM MENU ======")**

**print("1. Add Customer")**

**print("2. Remove Customer")**

**print("3. Add Vehicle")**

**print("4. View Available Vehicles")**

**print("5. Create Lease")**

**print("6. Return Car")**

**print("7. Record Payment")**

**print("8. View Active Leases")**

**print("9. View Lease History")**

**print("10. Exit")**

**choice = input("Enter your choice: ")**

**try:**

**if choice == "1":**

**first = input("First name: ")**

**last = input("Last name: ")**

**email = input("Email: ")**

**phone = input("Phone number: ")**

**customer = Customer(0, first, last, email, phone)**

**repo.addCustomer(customer)**

**elif choice == "2":**

**cid = int(input("Customer ID to remove: "))**

**repo.removeCustomer(cid)**

**elif choice == "3":**

**make = input("Make: ")**

**model = input("Model: ")**

**year = int(input("Year: "))**

**rate = float(input("Daily Rate: "))**

**status = "available"**

**capacity = int(input("Passenger Capacity: "))**

**engine = float(input("Engine Capacity: "))**

**vehicle = Vehicle(0, make, model, year, rate, status, capacity, engine)**

**repo.addCar(vehicle)**

**elif choice == "4":**

**available = repo.listAvailableCars()**

**print("\nAvailable Cars:")**

**for v in available:**

**print(v)**

**elif choice == "5":**

**cid = int(input("Customer ID: "))**

**vid = int(input("Vehicle ID: "))**

**start = input("Start date (YYYY-MM-DD): ")**

**end = input("End date (YYYY-MM-DD): ")**

**repo.createLease(cid, vid, start, end)**

**elif choice == "6":**

**lease\_id = int(input("Lease ID to return: "))**

**repo.returnCar(lease\_id)**

**elif choice == "7":**

**lease\_id = int(input("Lease ID: "))**

**amount = float(input("Payment Amount: "))**

**repo.recordPayment(lease\_id, amount)**

**elif choice == "8":**

**leases = repo.listActiveLeases()**

**print("\nActive Leases:")**

**for lease in leases:**

**print(lease)**

**elif choice == "9":**

**history = repo.listLeaseHistory()**

**print("\nLease History:")**

**for lease in history:**

**print(lease)**

**elif choice == "10":**

**print("Exiting... Goodbye!")**

**break**

**else:**

**print("Invalid option. Try again.")**

**except CustomerNotFoundException as e:**

**print(f"Error: {e}")**

**except VehicleNotFoundException as e:**

**print(f"Error: {e}")**

**except LeaseNotFoundException as e:**

**print(f"Error: {e}")**

**except Exception as e:**

**print(f"An unexpected error occurred: {e}")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

1. **reporting  
   8.1 generate\_test\_report.py**

**Purpose:**Generates an Excel report of unit-test outcomes using openpyxl.

**Code:**

**import openpyxl**

**from datetime import datetime**

**test\_results = [**

**("test\_add\_customer", "test\_customer.py", "Add a valid customer", "Customer added", "Customer added", "Pass"),**

**("test\_remove\_customer\_not\_found", "test\_customer.py", "Remove customer with invalid ID", "Raise Exception", "Exception raised", "Pass"),**

**("test\_create\_lease", "test\_lease.py", "Create lease for valid data", "Lease created", "Lease created", "Pass"),**

**("test\_record\_payment", "test\_payment.py", "Record lease payment", "Payment saved", "Payment saved", "Pass"),**

**("test\_find\_vehicle\_not\_found", "test\_vehicle.py", "Find non-existent vehicle", "Raise Exception", "Exception raised", "Pass"),**

**]**

**wb = openpyxl.Workbook()**

**ws = wb.active**

**ws.title = "Unit Test Report"**

**headers = ["Test Name", "Module", "Description", "Expected Outcome", "Actual Outcome", "Status"]**

**ws.append(headers)**

**for result in test\_results:**

**ws.append(result)**

**filename = f"Unit\_Test\_Report\_{datetime.today().strftime('%Y-%m-%d')}.xlsx"**

**wb.save(filename)**

**print(f"Report saved as {filename}")**