**CODE DOCUMENTATION**

**1. PROJECT OVERVIEW**

The **Car Rental System** is a Python-based console application for managing customers, vehicles, leases, and payments. It provides:

* **Customer Management:** Add, update, remove, list, and find customers.
* **Vehicle Management:** Add vehicles, change availability status, and list or find vehicles.
* **Lease Management:** Create daily or monthly leases (with cost calculation), return cars, and view active or past leases.
* **Payment Handling:** Record payments, view payment history, and calculate total revenue.
* **Reporting & Testing:** Generate an Excel report of unit-test results.
* **Persistence:** All data is stored in a MySQL database via a DAO layer.

**2. PROJECT STRUCTURE**

CarRentalSystem-CASESTUDY/

│ db.properties # Database connection settings

│ SQL\_SCRIPT.sql # DDL & sample DML

│ generate\_test\_report.py # Excel report script

│ README.md # Project overview & instructions

│

├───app/ # Application entry point

│ main.py

│

├───entity/ # Domain/model classes

│ ├── customer.py

│ ├── vehicle.py

│ ├── lease.py

│ └── payment.py

│

├───dao/ # Data-access abstraction & implementation

│ ├── icar\_lease\_repository.py

│ └── icar\_lease\_repository\_impl.py

│

├───exception/ # Custom exception classes

│ custom\_exceptions.py

│

├───util/ # Utility classes

│ ├── db\_property\_util.py

│ └── db\_conn\_util.py

│

└───tests/ # Unit tests (PyUnit)

├── test\_customer.py

├── test\_vehicle.py

├── test\_lease.py

└── test\_payment.py

**3. KEY PACKAGES AND MODULES**

**3.1 dao/**

* **ICarLeaseRepository.py**  
  Defines the abstract interface (using abc.ABC) for all CRUD and business operations:
  + Customer methods: addCustomer, removeCustomer, listCustomers, findCustomerById
  + Vehicle methods: addCar, removeCar, listAvailableCars, listRentedCars, findCarById
  + Lease methods: createLease, returnCar, listActiveLeases, listLeaseHistory
  + Payment method: recordPayment
* **ICarLeaseRepositoryImpl.py**  
  Implements the above interface with MySQL persistence. Key responsibilities:
  + Executes parameterized SQL statements
  + Commits transactions and closes connections in finally blocks
  + Raises custom exceptions (CustomerNotFoundException, etc.) on lookup failures

**3.2 entity/**

Encapsulates core data structures with no business logic—just attributes and \_\_str\_\_ methods:

* **customer.py**: Customer(customerID, firstName, lastName, email, phoneNumber)
* **vehicle.py**: Vehicle(vehicleID, make, model, year, dailyRate, status, passengerCapacity, engineCapacity)
* **lease.py**: Lease(leaseID, vehicleID, customerID, startDate, endDate, leaseType)
* **payment.py**: Payment(paymentID, leaseID, paymentDate, amount)

**3.3 exception/**

* **custom\_exceptions.py**  
  Defines CustomerNotFoundException, VehicleNotFoundException, and LeaseNotFoundException, each inheriting from Exception for clear error handling in the DAO and CLI layers.

**3.4 util/**

* **db\_property\_util.py**  
  Reads db.properties via configparser and returns connection parameters.
* **db\_conn\_util.py**  
  Uses the above parameters to create and return a MySQL Connection object.

**3.5 app/**

* **main.py**  
  Provides a menu-driven CLI that maps user selections to DAO calls. Catches and displays custom exceptions as user-friendly messages.

**3.6 tests/**

* **test\_customer.py**, **test\_vehicle.py**, **test\_lease.py**, **test\_payment.py**  
  Use Python’s unittest framework to verify both positive (“happy path”) and negative (“exception”) scenarios for every use case.

**3.7 Reporting Script**

* **generate\_test\_report.py**  
  Reads a hard-coded list of test outcomes and writes an Excel file (openpyxl) summarizing test name, expected vs. actual, and pass/fail status.

**4. DATABASE CONFIGURATION**

* **db.properties**

Stores MySQL connection settings:

host=localhost

user=root

password=your\_password

database=CarRentalDB

* **SQL\_SCRIPT.sql**

Contains CREATE DATABASE, CREATE TABLE statements for Vehicle, Customer, Lease, and Payment, plus INSERT statements with sample data.

**5. NOTABLE CONSIDERATIONS**

* **Layered Architecture**: Clear separation of concerns (entities, DAO, CLI).
* **Resource Management**: All database cursors and connections are closed in finally blocks to prevent leaks.
* **Exception Safety**: Custom “not found” exceptions are raised at the DAO level and handled in the CLI.
* **Test Coverage**: Unit tests verify CRUD operations and exception cases, ensuring reliability.
* **Reporting**: An Excel report provides a clear summary of test outcomes for stakeholders.