Car Rental System

Name: **Tejsinh J. Harale -Bhosale**

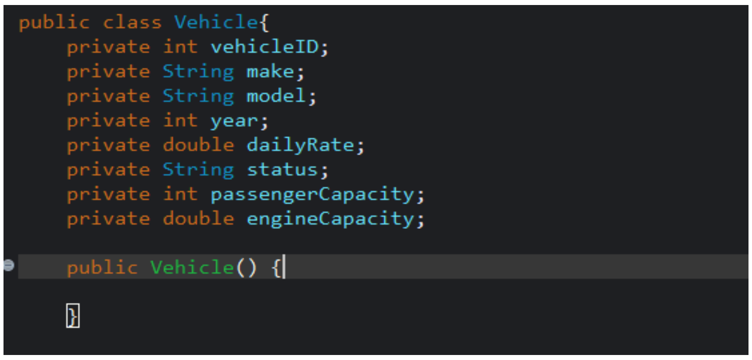
Github:**https://github.com/tejsinh3600/DA_foundation**

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:

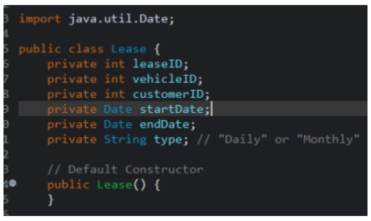
- vehicleID (Primary Key)
- make
- model
- year
- dailyRate
- status (available, notAvailable)
- passengerCapacity
- engineCapacity



```
public class Vehicle{
    private int vehicleID;
    private String make;
    private String model;
    private int year;
    private double dailyRate;
    private String status;
    private int passengerCapacity;
    private double engineCapacity;

    public Vehicle() {

    }
```

2. Customer Table:

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



```
import java.util.Date;

public class Lease {
    private int leaseID;
    private int vehicleID;
    private int customerID;
    private Date startDate;
    private Date endDate;
    private String type; // "Daily" or "Monthly"

    // Default Constructor
    public Lease() {
    }
```

3.Lease Table:

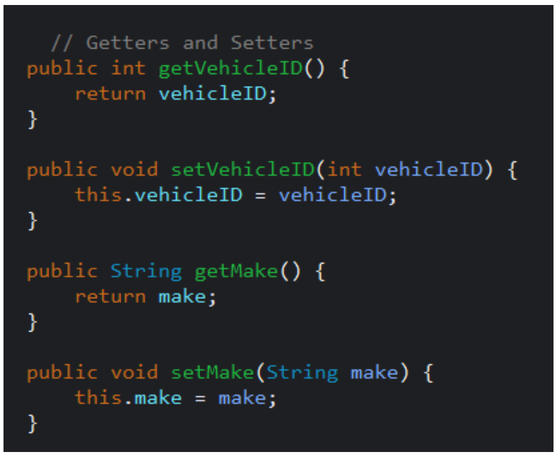
- leaseID (Primary Key)
- vehicleID (Foreign Key referencing Vehicle Table)
- customerID (Foreign Key referencing Customer Table)
- startDate
- endDate
- type (to distinguish between DailyLease and MonthlyLease)

4. Payment Table:

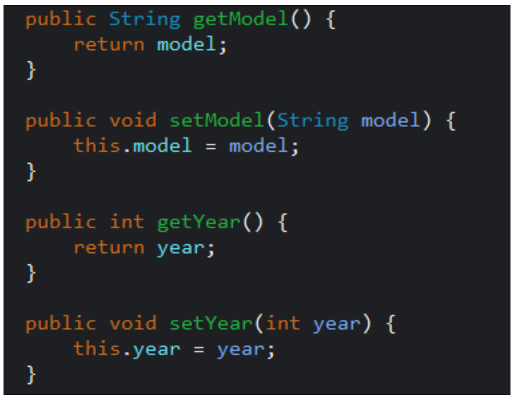
- paymentID (Primary Key)
- leaseID (Foreign Key referencing Lease Table)

- paymentDate
- amount

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



```java
    // Getters and Setters
    public int getVehicleID() {
        return vehicleID;
    }

    public void setVehicleID(int vehicleID) {
        this.vehicleID = vehicleID;
    }

    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        this.make = make;
    }
```

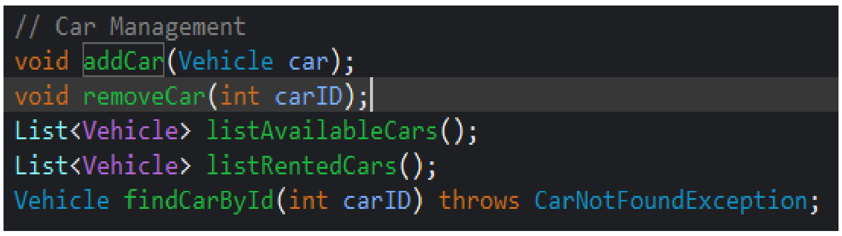


```java
    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }
```

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



```java
// Car Management
void addCar(Vehicle car);
void removeCar(int carID);
List<Vehicle> listAvailableCars();
List<Vehicle> listRentedCars();
Vehicle findCarById(int carID) throws CarNotFoundException;
```

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

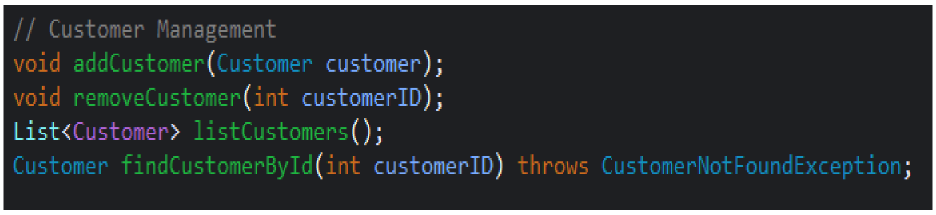
- parameter: NIL
- return type: return List of Car

5. findCarById(int carID) – return Car if found or throw exception

- parameter: NIL
- return type: return List of Car

- **Customer Management**

1. addCustomer(Customer customer)



```
// Customer Management
void addCustomer(Customer customer);
void removeCustomer(int customerID);
List<Customer> listCustomers();
Customer findCustomerById(int customerID) throws CustomerNotFoundException;
```

- 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)

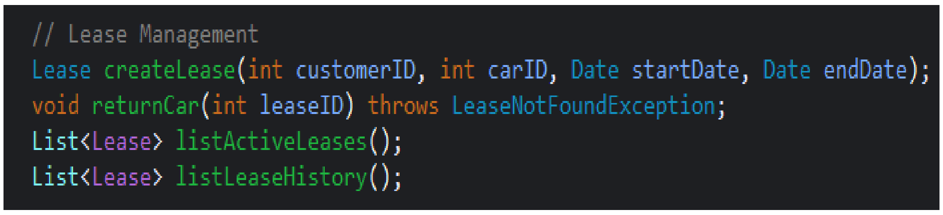
- parameter: CustomerID
- return type: Customer

- **Lease Management**

1. createLease()

- parameter: int customerID, int carID, Date startDate, Date endDate
- return type: Lease

2. void returnCar();



```
// Lease Management
Lease createLease(int customerID, int carID, Date startDate, Date endDate);
void returnCar(int leaseID) throws LeaseNotFoundException;
List<Lease> listActiveLeases();
List<Lease> listLeaseHistory();
```

- parameter: int leaseID
- return type: Lease info

3. List<Lease> listActiveLeases();

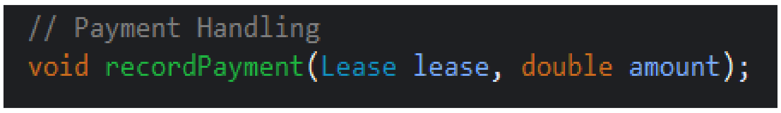
- parameter: NIL
- return type: Lease list

4. listLeaseHistory();

- parameter: NIL
- return type : Lease list

- **Payment Handling**

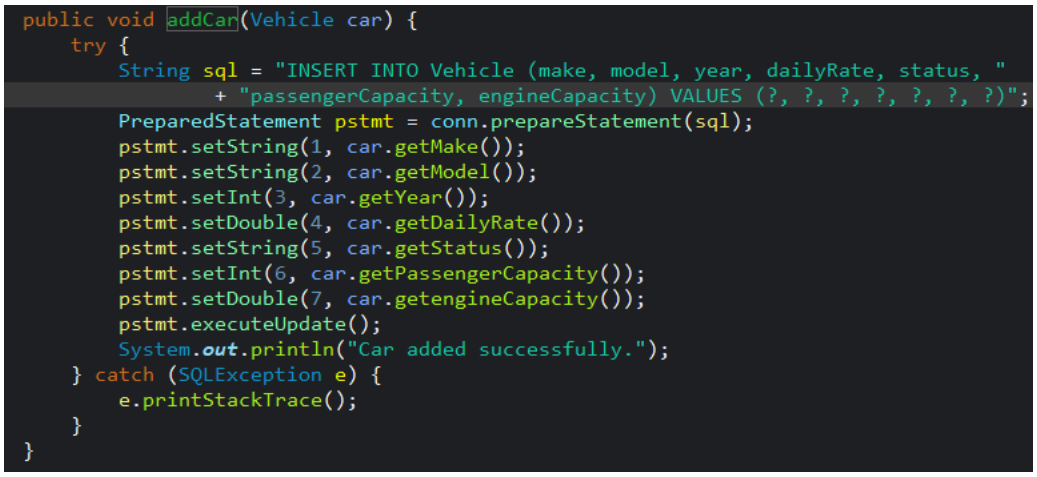
1. void recordPayment();



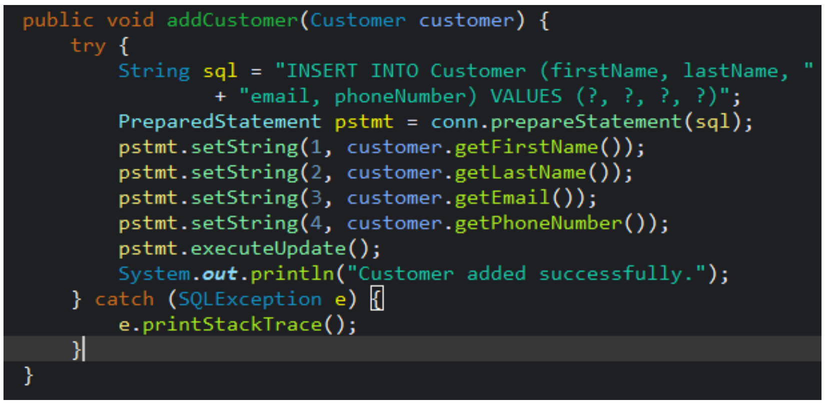
```
// Payment Handling
void recordPayment(Lease lease, double amount);
```

- parameter: Lease lease, double amount
- return type : void

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



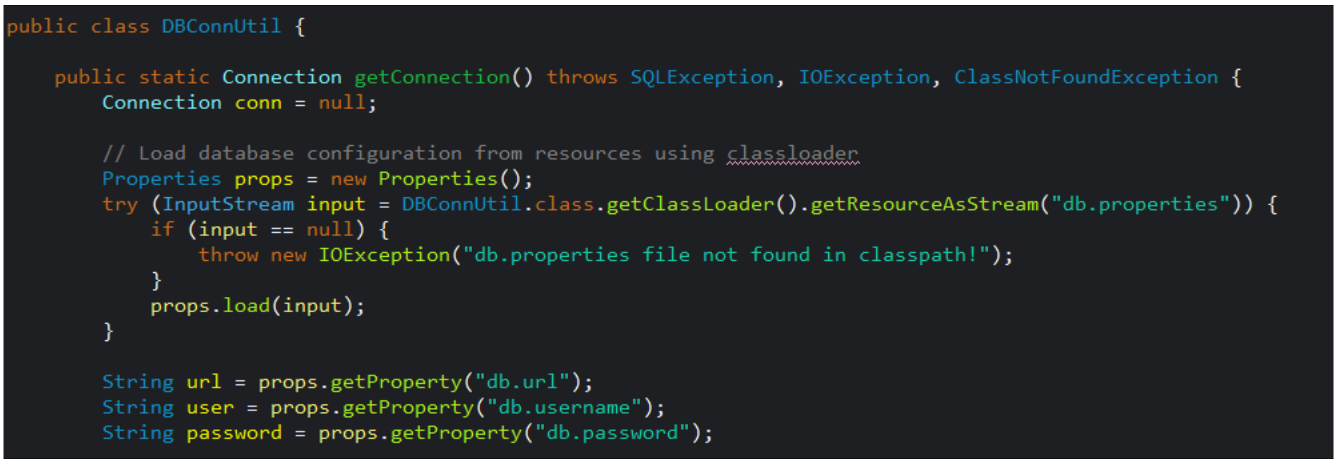
```java
public void addCar(Vehicle car) {
    try {
        String sql = "INSERT INTO Vehicle (make, model, year, dailyRate, status, "
                + "passengerCapacity, engineCapacity) VALUES (?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, car.getMake());
        pstmt.setString(2, car.getModel());
        pstmt.setInt(3, car.getYear());
        pstmt.setDouble(4, car.getDailyRate());
        pstmt.setString(5, car.getStatus());
        pstmt.setInt(6, car.getPassengerCapacity());
        pstmt.setDouble(7, car.getengineCapacity());
        pstmt.executeUpdate();
        System.out.println("Car added successfully.");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



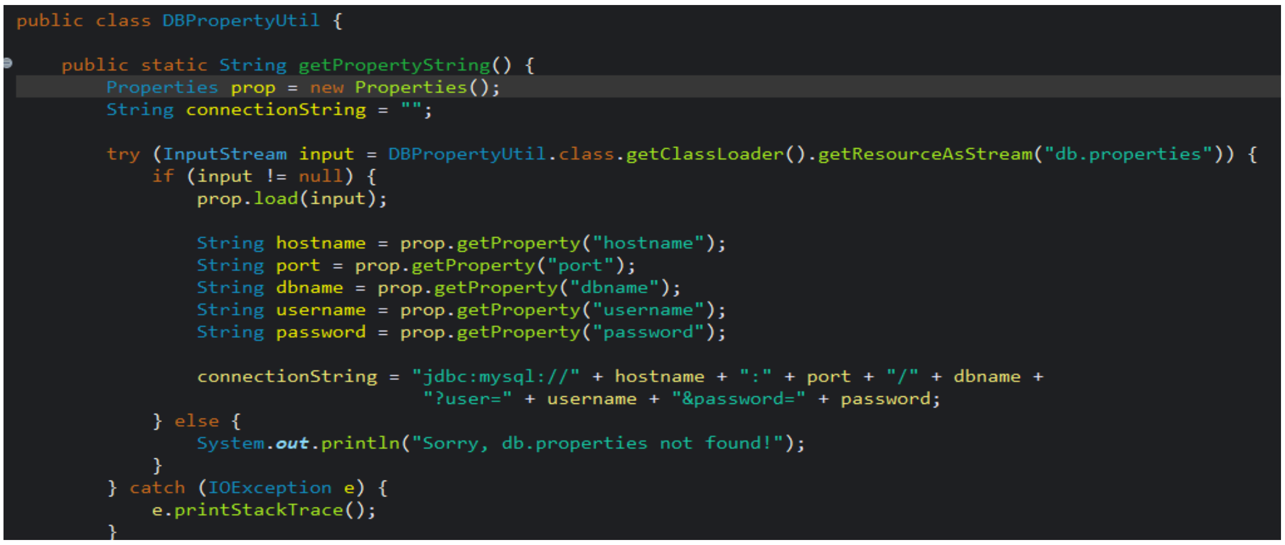
```java
public void addCustomer(Customer customer) {
    try {
        String sql = "INSERT INTO Customer (firstName, lastName, "
                + "email, phoneNumber) VALUES (?, ?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, customer.getFirstName());
        pstmt.setString(2, customer.getLastName());
        pstmt.setString(3, customer.getEmail());
        pstmt.setString(4, customer.getPhoneNumber());
        pstmt.executeUpdate();
        System.out.println("Customer added successfully.");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Connect your application to the SQL database:

8. Connect your application to the SQL database and write code to establish a connection to your SQL database.

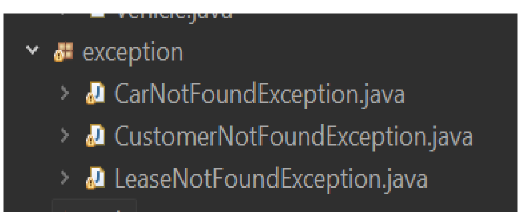


```java
public class DBConnUtil {

    public static Connection getConnection() throws SQLException, IOException, ClassNotFoundException {
        Connection conn = null;

        // Load database configuration from resources using classloader
        Properties props = new Properties();
        try (InputStream input = DBConnUtil.class.getClassLoader().getResourceAsStream("db.properties")) {
            if (input == null) {
                throw new IOException("db.properties file not found in classpath!");
            }
            props.load(input);
        }

        String url = props.getProperty("db.url");
        String user = props.getProperty("db.username");
        String password = props.getProperty("db.password");
```

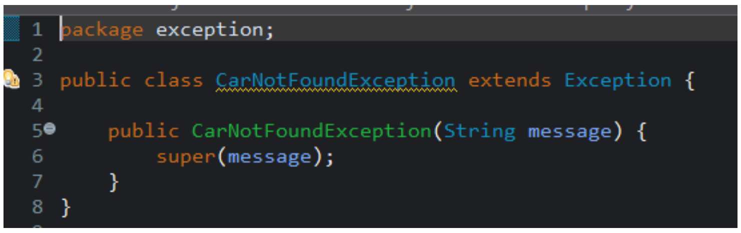


```java
public class DBPropertyUtil {

    public static String getPropertyString() {
        Properties prop = new Properties();
        String connectionString = "";

        try (InputStream input = DBPropertyUtil.class.getClassLoader().getResourceAsStream("db.properties")) {
            if (input != null) {
                prop.load(input);

                String hostname = prop.getProperty("hostname");
                String port = prop.getProperty("port");
                String dbname = prop.getProperty("dbname");
                String username = prop.getProperty("username");
                String password = prop.getProperty("password");

                connectionString = "jdbc:mysql://" + hostname + ":" + port + "/" + dbname +
                        "?user=" + username + "&password=" + password;
            } else {
                System.out.println("Sorry, db.properties not found!");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

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,



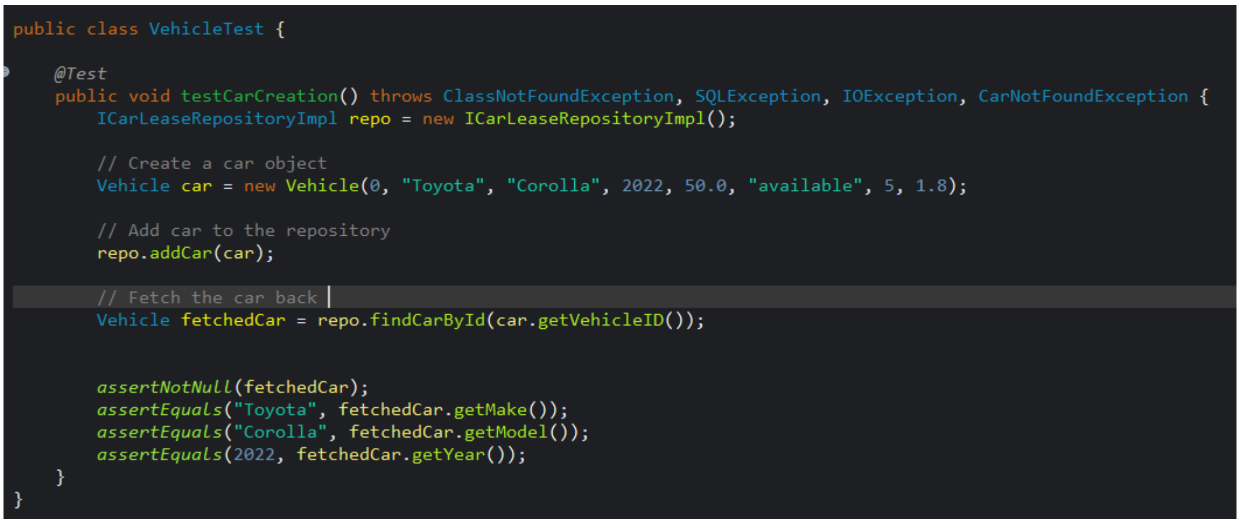
```
exception
    CarNotFoundException.java
    CustomerNotFoundException.java
    LeaseNotFoundException.java
```



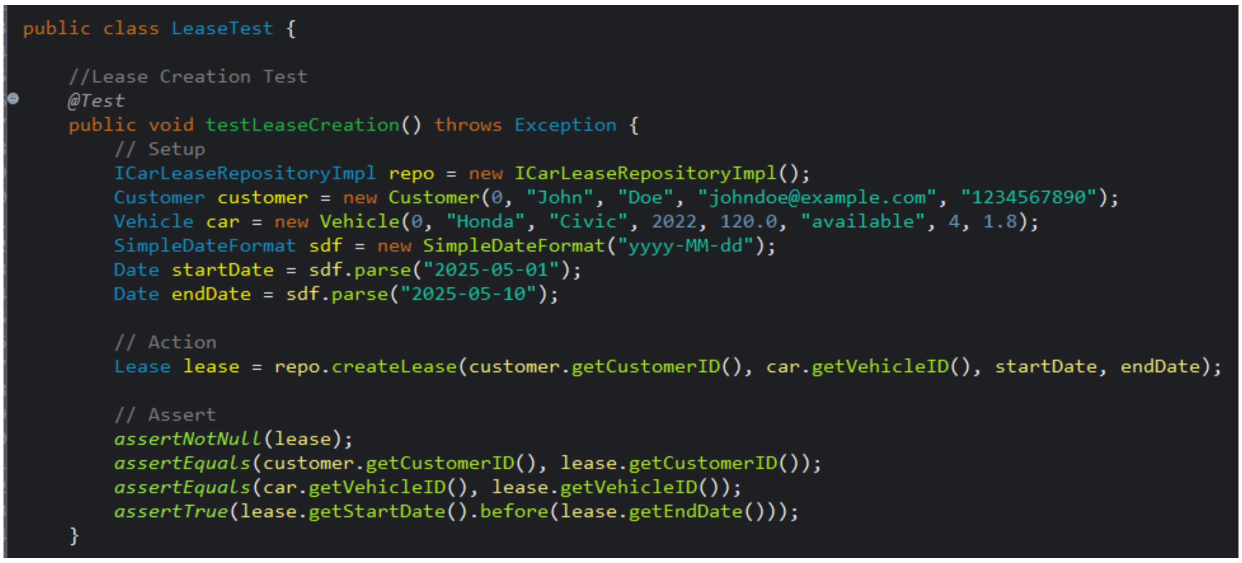
```java
package exception;

public class CarNotFoundException extends Exception {

    public CarNotFoundException(String message) {
        super(message);
    }
}
```

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:



```java
public class VehicleTest {

    @Test
    public void testCarCreation() throws ClassNotFoundException, SQLException, IOException, CarNotFoundException {
        ICarLeaseRepositoryImpl repo = new ICarLeaseRepositoryImpl();

        // Create a car object
        Vehicle car = new Vehicle(0, "Toyota", "Corolla", 2022, 50.0, "available", 5, 1.8);

        // Add car to the repository
        repo.addCar(car);

        // Fetch the car back
        Vehicle fetchedCar = repo.findCarById(car.getVehicleID());


        assertNotNull(fetchedCar);
        assertEquals("Toyota", fetchedCar.getMake());
        assertEquals("Corolla", fetchedCar.getModel());
        assertEquals(2022, fetchedCar.getYear());
    }
}
```



```java
public class LeaseTest {

    //Lease Creation Test
    @Test
    public void testLeaseCreation() throws Exception {
        // Setup
        ICarLeaseRepositoryImpl repo = new ICarLeaseRepositoryImpl();
        Customer customer = new Customer(0, "John", "Doe", "johndoe@example.com", "1234567890");
        Vehicle car = new Vehicle(0, "Honda", "Civic", 2022, 120.0, "available", 4, 1.8);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Date startDate = sdf.parse("2025-05-01");
        Date endDate = sdf.parse("2025-05-10");

        // Action
        Lease lease = repo.createLease(customer.getCustomerID(), car.getVehicleID(), startDate, endDate);

        // Assert
        assertNotNull(lease);
        assertEquals(customer.getCustomerID(), lease.getCustomerID());
        assertEquals(car.getVehicleID(), lease.getVehicleID());
        assertTrue(lease.getStartDate().before(lease.getEndDate()));
    }
```