Car Rental System

Name: **Atharva Padawale**

Github: https://github.com/AtharvaPadawale/hexaware-Java-Batch-5

Instructions

- Project submissions should be done through the partcipants' Github repository, and the link should be shared with trainers and Hexavarsity.
- Each section builds upon the previous one, and by the end, you will have a comprehensive Ecommerce implemented with a strong focus on SQL, control flow statements, loops, arrays, collections, exception handling, database interaction and Unit Testing.
- Follow object-oriented principles throughout the project. Use classes and objects to model realworld entities, encapsulate data and behavior, and ensure code reusability.
- Throw user defined exceptions from corresponding methods and handled.
- The following Directory structure is to be followed in the application.

entity/model

- Create entity classes in this package. All entity class should not have any business logic.

dao

• Create Service Provider interface to showcase functionalities.

• Create the implementation class for the above interface with db interaction.

exception

• Create user defined exceptions in this package and handle exceptions whenever needed.

util

• Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.

• Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object(Use method defined in DBPropertyUtil class to get the connection String).

main

• Create a class MainModule and demonstrate the functionalities in a menu driven application.

Key Functionalities:

1. Customer Management

- Add new customers, Update customer information, Retrieve customer details.

2. Car Management:

- Add new cars to the system, Update car availability, Retrieve car information.

3. Lease Management

- Create daily or monthly leases for customers.

- Calculate the total cost of a lease based on the type (Daily or Monthly) and the number of days or months.

4. Payment Handling:

- Record payments for leases.
- Retrieve payment history for a customer.
- Calculate the total revenue from payments.

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

```java
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

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)

```java
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() {
    }
```

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

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

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

• Customer Management

1. addCustomer(Customer customer)

- parameter: Customer
- return type: void

2. void removeCustomer(int customerID)

- parameter: CustomerID
- return type: void

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

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<Lease> listActiveLeases();

- parameter: NIL
- return type: Lease list

4. listLeaseHistory();

- parameter: NIL
- return type : Lease list

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

• Payment Handling

1. void recordPayment();

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

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

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

```
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.

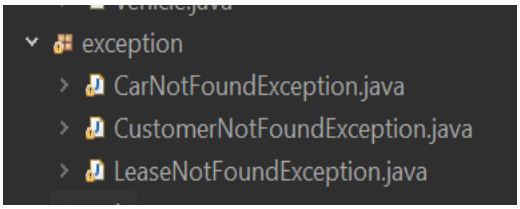
- Create a utility class DBConnection in a package util with a static variable connection of TypeConnection and a static method getConnection() which returns connection.
- Connection properties supplied in the connection string should be read from a property file.
- Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```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,

- CarNotFoundException: throw this exception when user enters an invalid car id which doesn't in db.
- LeaseNotFoundException: throw this exception when user enters an invalid lease id which exist in db.
- CustomerrNotFoundException: throw this exception when user enters an invalid customer id doesn't exist in db.




```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:

1. Write test case to test car created successfully or not.
2. Write test case to test lease is created successfully or not.
3. Write test case to test lease is retrieved successfully or not.
4. write test case to test exception is thrown correctly or not when customer id or car id or lease id not found in database.

```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()));
    }
```