

Web API Design with Spring Boot Week 3 Coding Assignment

Points possible: 70

Category	Criteria	% of Grade
Functionality	Does the code work?	25
Organization	Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear.	25
Creativity	Student solved the problems presented in the assignment using creativity and out of the box thinking.	25
Completeness	All requirements of the assignment are complete.	25

Instructions: In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed. Take screenshots of the code and of the running program (make sure to get screenshots of all required functionality) and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document, with your Java project code, to the repository. Add the URL for this week's repository to this document where instructed and submit this document to your instructor when complete.

Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon:  You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.

Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

- 1) In the application you've been building add a DAO layer:
 - a) Add the package, com.promineotech.jeepp.dao.
 - b) In the new package, create an interface named JeepSalesDao.
 - c) In the same package, create a class named DefaultJeepSalesDao that implements JeepSalesDao.
 - d) Add a method in the DAO interface and implementation that returns a list of Jeep models (class Jeep) and takes the model and trim parameters. Here is the method signature:

```
List<Jeep> fetchJeeps(JeepModel model, String trim);
```

- 2) In the Jeep sales service implementation class, inject the DAO interface as an instance variable. The instance variable should be private and should be named jeepSalesDao. Call the DAO method from the service method and store the returned value in a local variable named jeeps. Return the value in the jeeps variable (we will add to this later).
- 3) In the DAO implementation class (DefaultJeepSalesDao):
 - a) Add the class-level annotation: @Service.
 - b) Add a log statement in DefaultJeepSalesDao.fetchJeeps() that logs the model and trim level. Run the integration test. Produce a screenshot showing the DAO implementation class and the log line in the IDE's console.


The screenshot shows an IDE with two main panels. The top panel displays the source code of the `DefaultJeepSalesDao` class. The code is as follows:

```
1 package com.promineotech.jeep.dao;
2
3 import java.util.List;
4
5 @Component
6 @Service
7 public class DefaultJeepSalesDao implements JeepSalesDao {
8
9     @Override
10    public List<Jeep> fetchJeeps(JeepModel model, String trim) {
11        log.debug("DAO: model={}, trim={}", model, trim);
12        return null;
13    }
14 }
```

The bottom panel shows the console output, which includes the following log line:

```
16:09:59.816 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Before test class: context [DefaultTestContext@4d4e02fc testClass = FetchJeepTest, testInstance = com.promineotech.jeep.dao.DefaultJeepSalesDao, testMethod = null, testArgs = null]
```


- c)
- d) In `DefaultJeepSalesDao`, inject an instance variable of type `NamedParameterJdbcTemplate`.

- e) Write SQL to return a list of Jeep models based on the parameters: model and trim. Be sure to utilize the SQL Injection prevention mechanism of the NamedParameterJdbcTemplate using :model_id and :trim_level in the query.
- f) Add the parameters to a parameter map as shown in the video. Don't forget to convert the JeepModel enum value to a String (i.e., params.put("model_id", model.toString());)
- g) Call the query method on the NamedParameterJdbcTemplate instance variable to return a list of Jeep model objects. Use a RowMapper to map each row of the result set. Remember to convert modelId to a JeepModel. See the video for details. Produce a screenshot to show the complete method in the implementation class. 

```

1 package com.promineotech.jeep.dao;
2
3 import java.math.BigDecimal;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import java.util.HashMap;
7 import java.util.List;
8 import java.util.Map;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.jdbc.core.RowMapper;
11 import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
12 import org.springframework.stereotype.Component;
13 import com.promineotech.jeep.entity.Jeep;
14 import com.promineotech.jeep.entity.JeepModel;
15 import lombok.extern.slf4j.Slf4j;
16
17 @Component
18 @Slf4j
19 public class DefaultJeepSalesDao implements JeepSalesDao {
20
21     @Autowired
22     private NamedParameterJdbcTemplate jdbcTemplate;
23
24     @Override
25     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
26         Log.debug("DAO: model = {}, trim = {}", model, trim);
27
28         String sql = "" + "SELECT * " + "FROM models " + "WHERE model_id = :model_id AND trim_level = :trim_level";
29
30         Map<String, Object> params = new HashMap<>();
31         params.put("model_id", model.toString());
32         params.put("trim_level", trim);
33
34         return jdbcTemplate.query(sql, params, new RowMapper<>() {
35             @Override
36             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
37                 return Jeep.builder()
38                     .basePrice(new BigDecimal(rs.getString("base_price")))
39                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
40                     .modelPK(rs.getLong("model_pk"))
41                     .numDoors(rs.getInt("num_doors"))
42                     .trimLevel(rs.getString("trim_level"))
43                     .wheelSize(rs.getInt("wheel_size"))
44                     .build();
45             }
46         });
47     }
48
49 }

```

- 4) Add a getter in the Jeep class for modelPK. Add the @JsonIgnore annotation to the getter to exclude the modelPK value from the returned object.
- 5) Run the test to produce a green status bar. Produce a screenshot showing the test and the green status bar. 

```
19 import org.springframework.test.context.jdbc.SqlConfig;
20 import com.prowinotech.jee.entity.Jee;
21 import com.prowinotech.jee.entity.JeeModel;
22
23 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
24 @ActiveProfiles("test")
25 @Sql(scripts = {
26     "classpath:flyway/migrations/V1_0_Jee_Schema.sql",
27     "classpath:flyway/migrations/V1_1_Jee_Data.sql"},
28     config = @SqlConfig(encoding = "utf-8"))
29
30 class FetchJeeTest {
31
32     @Test
33     void testThatJeeIsReturnedWhenValidModelAndTrimAreSupplied() {
34
35         //Given: A valid model, trim and URL
36         JeeModel model = JeeModel.WRANGLER;
37         String trim = "Sport";
38         String url = String.format("http://localhost:%d/jee/model=%s&trim=%s", serverPort, model, trim);
39
40         //When: A connection is made to the URI
41         ResponseEntity<List<Jee>> response = restTemplate.exchange(url, HttpMethod.GET, null, new ParameterizedTypeReference<>());
42
43         //Then: A success (OK - 200) status code is returned
44         assertEquals(response.getStatusCode(), HttpStatus.OK);
45
46         //And: The actual list returned is the same as the expected list
47         List<Jee> actual = response.getBody();
48         List<Jee> expected = buildExpected();
49         assertEquals(actual, expected);
50     }
51
52     private List<Jee> buildExpected() {
53         List<Jee> list = new LinkedList<>();
54
55         list.add(jee.builder()
56             .modelId(JeeModel.WRANGLER)
57             .trimLevel("Sport")
58             .numDoors(2)
59             .wheelSize(17)
60             .basePrice(new BigDecimal("28475.00"))
61             .build());
62
63         list.add(jee.builder()
64             .modelId(JeeModel.WRANGLER)
65             .trimLevel("Sport")
66             .numDoors(4)
67             .wheelSize(17)
68             .basePrice(new BigDecimal("33975.00"))
69             .build());
70
71         return list;
72     }
73
74     @Autowired
75     private TestRestTemplate restTemplate;
76
77     @LocalServerPort
78     private int serverPort;
79
80
81 }

```

Console

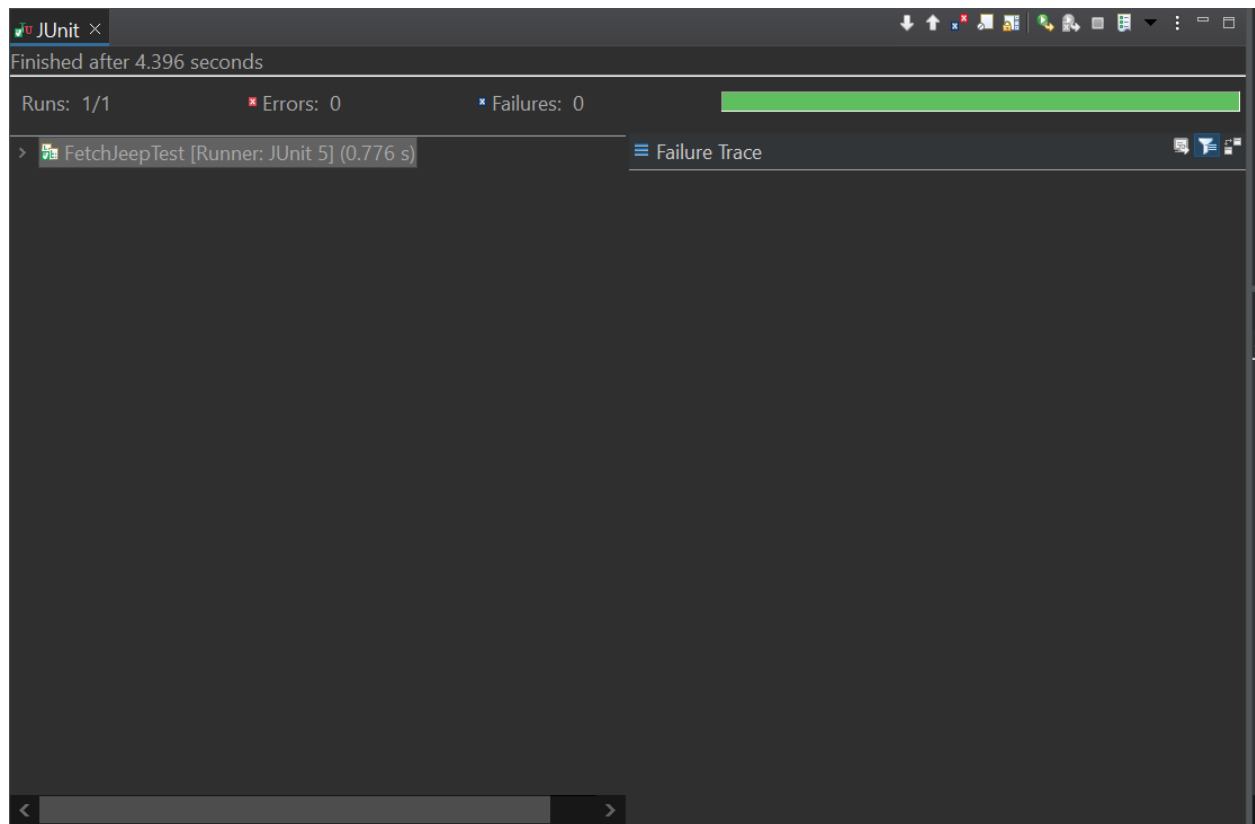
```

terminated> FetchJeeTest [Junit] C:\Users\thepo\workspace\spring-tool-suite-4-4.1\RELEASE\plugins\org.eclipse.jdt.launcher\org.eclipse.jdt.launcher.exe (Aug 10, 2022, 4:35:03 PM - 4:35:09 PM) [pid: 41744]

Spring
=====
:: Spring Boot ::
(v2.7.2)

2022-08-10 16:35:05.261 INFO 41744 --- [main] c.p.jee.controller.FetchJeeTest : Starting FetchJeeTest using Java 17.0.3 on DESKTOP-HQ050E1 with PID 41744 (started by thepo in C:\Users\thepo\workspace-spring-tool-suite-4-4.1)
2022-08-10 16:35:05.262 DEBUG 41744 --- [main] c.p.jee.controller.FetchJeeTest : Running with Spring Boot v2.7.2, Spring v5.3.22
2022-08-10 16:35:05.283 INFO 41744 --- [main] c.p.jee.controller.FetchJeeTest : The following 'profile' is active: 'test'
2022-08-10 16:35:08.121 INFO 41744 --- [main] c.p.jee.controller.FetchJeeTest : Started FetchJeeTest in 3.181 seconds (JVM running for 4.822)
2022-08-10 16:35:08.728 DEBUG 41744 --- [o-auto-1-exec-2] c.p.j.c.DefaultJeeSalesController : model=WRANGLER, trim=Sport
2022-08-10 16:35:08.728 INFO 41744 --- [o-auto-1-exec-2] c.p.j.service.DefaultJeeSalesService : The fetchJeeps method was called with model=WRANGLER and trim=Sport
2022-08-10 16:35:08.728 DEBUG 41744 --- [o-auto-1-exec-2] c.p.jee.dao.DefaultJeeSalesDao : DAO: model = WRANGLER, trim = Sport

```



URL to GitHub Repository:

<https://github.com/coltonrood/week15>