

# 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.

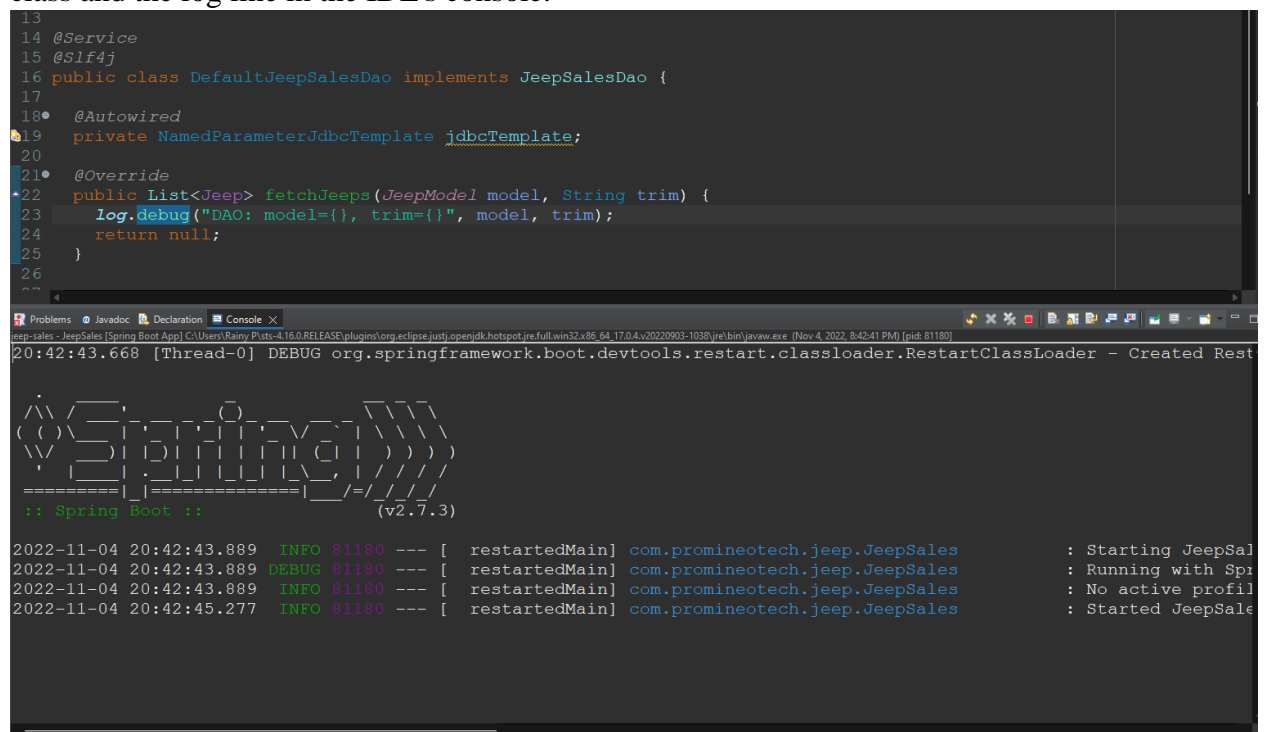
**Project Resources:** <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

## 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 panels. The top panel displays the `DefaultJeepSalesDao` class, which implements `JeepSalesDao`. It includes annotations `@Service` and `@Slf4j`, and a private field `NamedParameterJdbcTemplate jdbcTemplate`. The `fetchJeeps` method is overridden and contains a debug log statement: `log.debug("DAO: model={}, trim={}", model, trim);` followed by `return null;`. The bottom panel shows the IDE's console output, which includes a message from Spring Boot (v2.7.3) and several log entries from the application. The logs show the application starting, running with Spring, and the `fetchJeeps` method being called with model and trim parameters.


```
13
14 @Service
15 @Slf4j
16 public class DefaultJeepSalesDao implements JeepSalesDao {
17
18     @Autowired
19     private NamedParameterJdbcTemplate jdbcTemplate;
20
21     @Override
22     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
23         log.debug("DAO: model={}, trim={}", model, trim);
24         return null;
25     }
26
27 }
```

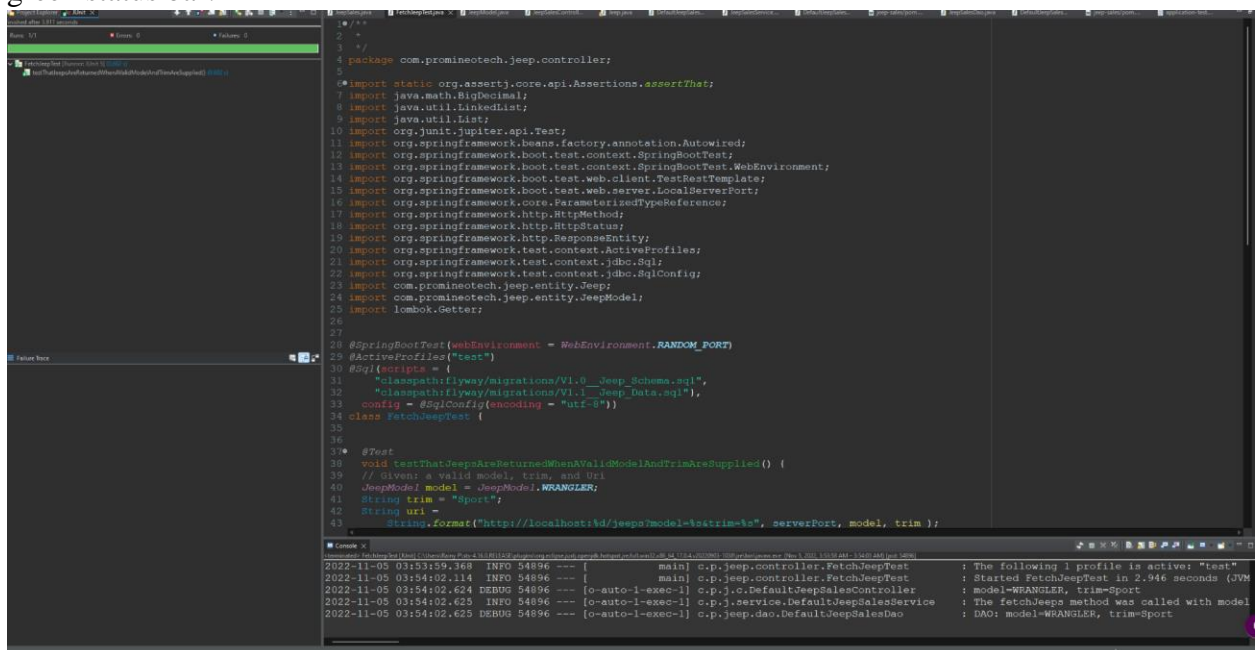
```
20:42:43.668 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created Rest
:: Spring Boot :: (v2.7.3)
2022-11-04 20:42:43.889 INFO 81180 --- [ restartedMain] com.promineotech.jeep.JeepSales : Starting JeepSal
2022-11-04 20:42:43.889 DEBUG 81180 --- [ restartedMain] com.promineotech.jeep.JeepSales : Running with Spr
2022-11-04 20:42:43.889 INFO 81180 --- [ restartedMain] com.promineotech.jeep.JeepSales : No active profil
2022-11-04 20:42:45.277 INFO 81180 --- [ restartedMain] com.promineotech.jeep.JeepSales : Started JeepSale
```

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

```
13 import com.promineotech.jee.entity.Jee;
14 import com.promineotech.jee.entity.JeeModel;
15 import lombok.extern.slf4j.Slf4j;
16
17 @Component
18 @Slf4j
19 public class DefaultJeeSalesDao implements JeeSalesDao {
20
21     @Autowired
22     private NamedParameterJdbcTemplate jdbcTemplate;
23
24     @Override
25     public List<Jee> fetchJeeps(JeeModel model, String trim) {
26         log.debug("DAO: model={}, trim={}", model, trim);
27
28         //formatter:off
29         String sql = "
30             + "SELECT * "
31             + "FROM models "
32             + "WHERE model_id = :model_id AND trim_level = :trim_level";
33         //formatter:on
34
35         Map<String, Object> params = new HashMap<>();
36         params.put("model_id", model.toString());
37         params.put("trim_level", trim);
38
39         return jdbcTemplate.query(sql, params,
40             new RowMapper<>() {
41                 @Override
42                 public Jee mapRow(ResultSet rs, int rowNum) throws SQLException {
43                     //formatter:off
44                     return Jee.builder()
45                         .basePrice(new BigDecimal(rs.getString("base_price")))
46                         .modelId(JeeModel.valueOf(rs.getString("model_id")))
47                         .modelPK(rs.getLong("model_pk"))
48                         .numDoors(rs.getInt("num_doors"))
49                         .trimLevel(rs.getString("trim_level"))
50                         .wheelSize(rs.getInt("wheel_size"))
51                         .build();
52                     //formatter:on
53                 }
54             })
55     }
56 }
```

- 4) Add a getter in the Jee 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. 



The screenshot shows an IDE with a Java test class `FetchJeeTest` and its execution output. The test class is located in `com.promineotech.jee.controller` and imports various dependencies including `org.assertj.core.api.Assertions`, `java.math.BigDecimal`, `java.util.LinkedList`, `java.util.List`, `org.junit.jupiter.api.Test`, `org.springframework.beans.factory.annotation.Autowired`, `org.springframework.boot.test.context.SpringBootTest`, `org.springframework.boot.test.web.client.TestRestTemplate`, `org.springframework.boot.test.web.server.LocalServerPort`, `org.springframework.core.ParameterizedTypeReference`, `org.springframework.http.HttpStatus`, `org.springframework.http.ResponseEntity`, `org.springframework.test.context.ActiveProfiles`, `org.springframework.test.context.jdbc.Sql`, `org.springframework.test.context.jdbc.SqlConfig`, `com.promineotech.jee.entity.Jee`, `com.promineotech.jee.entity.JeeModel`, and `lombok.Getter`.

The test class includes a `@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)` annotation, a `@Sql(scripts = { "classpath:flyway/migrations/V1.0_Jee.Schema.sql", "classpath:flyway/migrations/V1.1_Jee.Data.sql" })` annotation, and a `@SqlConfig(encoding = "utf-8")` annotation. The test method `testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()` is annotated with `@Test` and `@ActiveProfiles("test")`. It sets up a `JeeModel` with `model = JeeModel.WRANGLER` and `trim = "Sport"`, and then calls the `fetchJeeps` method of the `DefaultJeeSalesDao` using a `TestRestTemplate`.

The execution output shows the test passing successfully. The output includes the following lines:

```
2022-11-05 03:53:59.456 INFO 54896 --- [main] c.p.jee.controller.FetchJeeTest : The following 1 profile is active: "test"
2022-11-05 03:54:02.114 INFO 54896 --- [main] c.p.jee.controller.FetchJeeTest : Started FetchJeeTest in 2.946 seconds (JVM
2022-11-05 03:54:02.624 DEBUG 54896 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeeSalesController : model=WRANGLER, trim=Sport
2022-11-05 03:54:02.625 INFO 54896 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeeSalesService : The fetchJeeps method was called with model
2022-11-05 03:54:02.625 DEBUG 54896 --- [o-auto-1-exec-1] c.p.jee.dao.DefaultJeeSalesDao : DAO: model=WRANGLER, trim=Sport
```

Screenshots of Code:

**Screenshots of Running Application:**

**URL to GitHub Repository:**

[Upload files · RPador/Promineo-Week-15 \(github.com\)](#)