


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`):
- Add the class-level annotation: `@Service`.
 - 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. 

```

3 import java.math.BigDecimal;
8
9 @Component
0 @Slf4j
1 public class DefaultJeepSalesDao implements JeepSalesDao {
2
3     @Autowired
4     private NamedParameterJdbcTemplate jdbcTemplate;
5
6     @Override
7     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
8         log.debug("DAO: model={}, trim={}", model, trim);
9
10        // @formatter: off
11        String sql = ""
12            + "SELECT * "
13            + "FROM models "
14            + "WHERE model_id = :model_id AND trim_level = :trim_level";
15        // @formatter: on
16
17        Map<String, Object> params = new HashMap<>();
18        params.put("model_id", model.toString());
19        params.put("trim_level", trim);
20
21        return jdbcTemplate.query(sql, params,
22            new RowMapper<>() {
23
24            @Override
25            public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
26                // formatter:off
27                return Jeep.builder()
28                    .basePrice(new BigDecimal(rs.getString("base_price"))
29                    .modelId(JeepModel.valueOf(rs.getString("model_id"))
30                    .modelPK(rs.getLong("model_pk"))
31                    .numDoors(rs.getInt("num_doors"))
32                    .trimLevel(rs.getString("trim_level"))
33                    .wheelSize(rs.getInt("wheel_size"))
34                    .build();
35                // formatter:on
36            }
37        });
38    }
39 }

```

```

.498 DEBUG 8067 --- [o-auto-1-exec-1] c.p.j.c.BasicJeepSalesController : model=WRANGLER, trim=Sport
.500 INFO 8067 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called
.500 DEBUG 8067 --- [o-auto-1-exec-1] c.p.jeeop.dao.DefaultJeepSalesDao : DAO: model=WRANGLER, trim=Sport

```

- 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. 🖥️

```
// @formatter: on

Map<String, Object> params = new HashMap<>();
params.put("model_id", model.toString());
params.put("trim_level", trim);

return jdbcTemplate.query(sql, params,
    new RowMapper<>() {

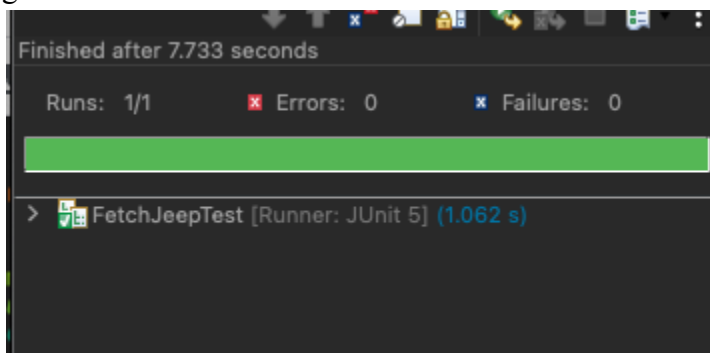
        @Override
        public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
            // formatter:off
            return Jeep.builder()
                .basePrice(new BigDecimal(rs.getString("base_price"))
                .modelId(JeepModel.valueOf(rs.getString("model_id"))
                .modelPK(rs.getLong("model_pk"))
                .numDoors(rs.getInt("num_doors"))
                .trimLevel(rs.getString("trim_level"))
                .wheelSize(rs.getInt("wheel_size"))
                .build();
            // formatter:on
        }
    });
}
```

- 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

```
@JsonIgnore
public Long getModelPK() {
    return modelPK;
}

@Override
public int compareTo(Jeep that) {
```

green status bar. 🖥️



Screenshots of Code: see above

Screenshots of Running Application:

```

23:38:58.236 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from class [org.springframework
23:38:58.254 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [public org.springframework
23:38:58.318 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test class [com.promineotech.jee
23:38:58.334 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Neither @ContextConfiguration nor @ContextHierarchy f
23:38:58.338 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test class [com
23:38:58.338 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test class [com
23:38:58.338 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not detect default configuration classes fo
23:38:58.338 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not detect default configuration classes fo
23:38:58.539 [main] DEBUG org.springframework.test.context.annotation.ClassPathScanningCandidateComponentProvider - Identified candidate component class: fil
23:38:58.542 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Found @SpringBootConfiguration com.promineotech.jee
23:38:58.635 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper - @TestExecutionListeners is not present for class [co
23:38:58.635 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Loaded default TestExecutionListener class names from
23:38:58.650 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.t
23:38:58.654 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTest
23:38:58.672 [main] DEBUG org.springframework.test.context.support.DependencyInjectionTestExecutionListener - Performing dependency injection for test c

```



```

:: Spring Boot ::
                (v2.6.7)

2022-05-31 23:38:59.261 INFO 8067 --- [           main] c.p.jee.controller.FetchJeeTest      : Starting FetchJeeTest using Java 17.0.2 on chelsea
2022-05-31 23:38:59.262 DEBUG 8067 --- [           main] c.p.jee.controller.FetchJeeTest      : Running with Spring Boot v2.6.7, Spring v5.3.19
2022-05-31 23:38:59.262 INFO 8067 --- [           main] c.p.jee.controller.FetchJeeTest      : The following 1 profile is active: "test"
2022-05-31 23:39:04.586 INFO 8067 --- [           main] c.p.jee.controller.FetchJeeTest      : Started FetchJeeTest in 5.874 seconds (JVM running
http://localhost:51292/jee?model=WRANGLER&trim=Sport
2022-05-31 23:39:05.498 DEBUG 8067 --- [o-auto-1-exec-1] c.p.j.c.BasicJeeSalesController      : model=WRANGLER, trim=Sport
2022-05-31 23:39:05.500 INFO 8067 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeeSalesService : The fetchJeeps method was called with model=WRANGLER
2022-05-31 23:39:05.500 DEBUG 8067 --- [o-auto-1-exec-1] c.p.jee.dao.DefaultJeeSalesDao       : DAO: model=WRANGLER, trim=Sport

```

URL to GitHub Repository: