


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.

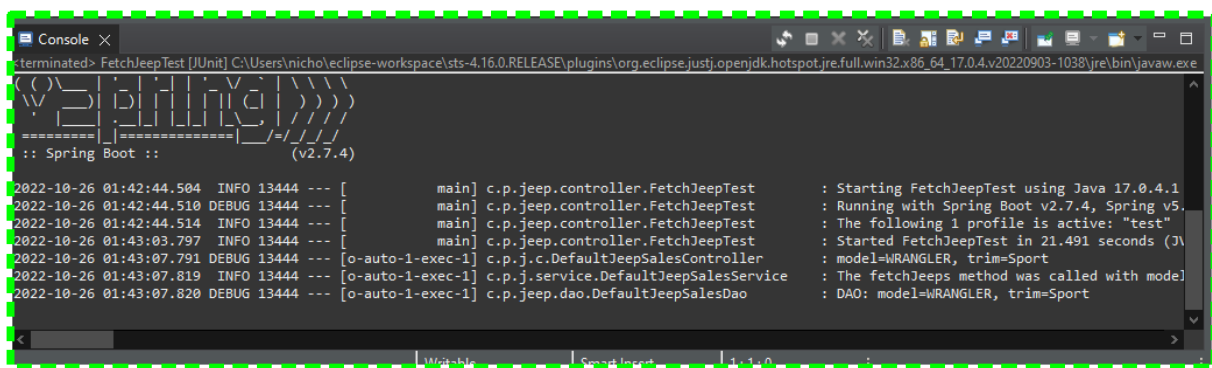
Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

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. 🖥️



```
terminated> FetchJeepTest [JUnit] C:\Users\nicho\workspace\sts-4.16.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1030\jre\bin\javaw.exe
:: Spring Boot :: (v2.7.4)
2022-10-26 01:42:44.504 INFO 13444 --- [main] c.p.jeepp.controller.FetchJeepTest : Starting FetchJeepTest using Java 17.0.4.1
2022-10-26 01:42:44.510 DEBUG 13444 --- [main] c.p.jeepp.controller.FetchJeepTest : Running with Spring Boot v2.7.4, Spring v5.
2022-10-26 01:42:44.514 INFO 13444 --- [main] c.p.jeepp.controller.FetchJeepTest : The following 1 profile is active: "test"
2022-10-26 01:43:03.797 INFO 13444 --- [main] c.p.jeepp.controller.FetchJeepTest : Started FetchJeepTest in 21.491 seconds (J
2022-10-26 01:43:07.791 DEBUG 13444 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : model=WRANGLER, trim=Sport
2022-10-26 01:43:07.819 INFO 13444 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepSalesService : The fetchJeeps method was called with model
2022-10-26 01:43:07.820 DEBUG 13444 --- [o-auto-1-exec-1] c.p.jeepp.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. 🖥️

```

@Override
public List<Jeep> fetchJeeps(JeepModel model, String trim) {
    Log.debug("DAO: model={}, trim={}", model, trim);


    // @formatter: off
    String sql = ""
        + "SELECT * "
        + "FROM models "
        + "WHERE model_id = :model_id AND trim_level = :trim_level";
    // @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 green status bar. 

Screenshots of Code:

```
JeepSalesDao.java × DefaultJeepSalesDao.java FetchJeepTest.java
1 package com.promineotech.jeep.dao;
2
3 import java.util.List;
4
5 import com.promineotech.jeep.entity.Jeep;
6 import com.promineotech.jeep.entity.JeepModel;
7
8 public interface JeepSalesDao {
9
10     /**
11      *
12      * @param model
13      * @param trim
14      * @return
15      */
16
17     List<Jeep> fetchJeeps(JeepModel model, String trim);
18
19 }
20
21
```

```
DefaultJeepSalesDao.java × FetchJeepTest.java DefaultJeepSalesService.java
19
20 @Component
21 @Slf4j
22 public class DefaultJeepSalesDao implements JeepSalesDao {
23
24     @Autowired
25     private NamedParameterJdbcTemplate jdbcTemplate;
26
27     @Override
28     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
29         log.debug("DAO: model={}, trim={}", model, trim);
30
31         // @formatter: off
32         String sql = ""
33             + "SELECT * "
34             + "FROM models "
35             + "WHERE model_id = :model_id AND trim_level = :trim_level";
36         // @formatter: on
37
38         Map<String, Object> params = new HashMap<>();
39         params.put("model_id", model.toString());
40         params.put("trim_level", trim);
41
42         return jdbcTemplate.query(sql, params,
43             new RowMapper<>() {
44
45                 @Override
46                 public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
47                     // @formatter: off
48                     return Jeep.builder()
49                         .basePrice(new BigDecimal(rs.getString("base_price")))
50                         .modelId(JeepModel.valueOf(rs.getString("model_id")))
51                         .modelPK(rs.getLong("model_pk"))
52                         .numDoors(rs.getInt("num_doors"))
53                         .trimLevel(rs.getString("trim_level"))
54                         .wheelSize(rs.getInt("wheel_size"))
55                         .build();
56                     // @formatter: on
57                 }
58             });
59     }
60 }
```

```
FetchJeepTest.java  Jeep.java  DefaultJeepSalesService.j
6  import com.fasterxml.jackson.annotation.JsonIgnore;
7
8  import lombok.AllArgsConstructor;
9  import lombok.Builder;
10 import lombok.Data;
11 import lombok.NoArgsConstructor;
12
13
14 @Data
15 @Builder
16 @NoArgsConstructor
17 @AllArgsConstructor
18 public class Jeep implements Comparable<Jeep> {
19     private Long modelPK;
20     private JeepModel modelId;
21     private String trimLevel;
22     private int numDoors;
23     private int wheelSize;
24     private BigDecimal basePrice;
25
26     @JsonIgnore
27     private Long getModelPk() {
28         return modelPK;
29     }
30
31     @Override
32     public int compareTo(Jeep that) {
33         // @formatter: off
34         return Comparator
35             .comparing(Jeep::getModelId)
36             .thenComparing(Jeep::getTrimLevel)
37             .thenComparing(Jeep::getNumDoors)
38             .compare(this, that);
39         // @formatter: on
40     }
41
42
43 } //end of class
44
```

```

FetchJeepTest.java × DefaultJeepSalesService.java ×
1 package com.promineotech.jeepp.service;
2
3 import java.util.Collections;
14
15 @Service
16 @Slf4j
17 public class DefaultJeepSalesService implements JeepSalesService {
18
19     @Autowired
20     private JeepSalesDao jeepSalesDao;
21
22     @Override
23     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
24         log.info("The fetchJeeps method was called with model={} and trim={}", model, trim);
25
26         List<Jeep> jeeps = jeepSalesDao.fetchJeeps(model, trim);
27
28         Collections.sort(jeeps);
29         |
30         return jeeps;
31     }
32
33 }
34

```

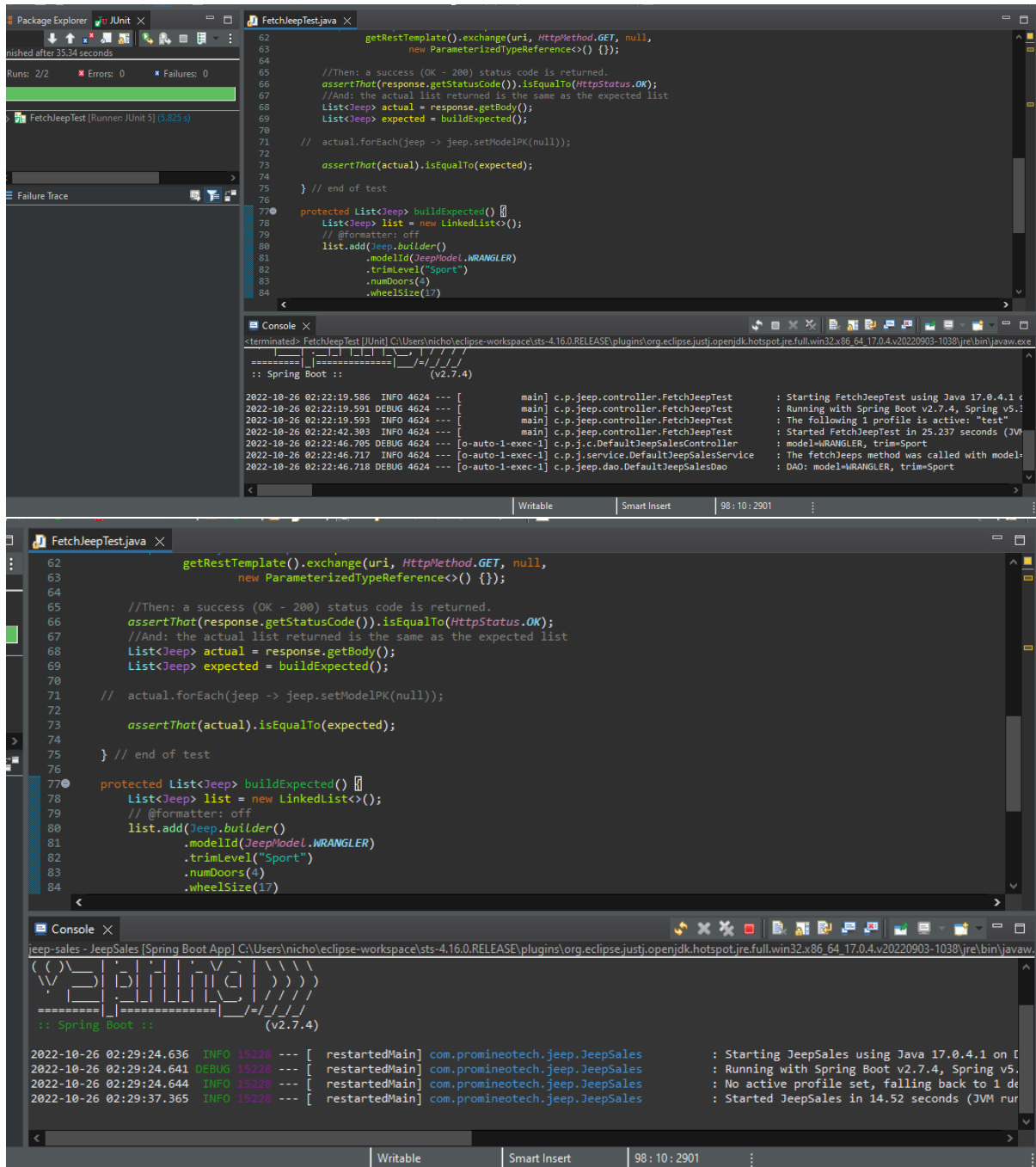
```

FetchJeepTest.java ×
48 void testDb() {
49
50 }
51
52
53 @Test
54 void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
55     //Given: a valid model, trim, and URI
56     JeepModel model = JeepModel.WRANGLER;
57     String trim = "Sport";
58     String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
59
60     //When: a connection is made to the URI
61     ResponseEntity<List<Jeep>> response =
62         getRestTemplate().exchange(uri, HttpMethod.GET, null,
63             new ParameterizedTypeReference<>() {});
64
65     //Then: a success (OK - 200) status code is returned.
66     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
67     //And: the actual list returned is the same as the expected list
68     List<Jeep> actual = response.getBody();
69     List<Jeep> expected = buildExpected();
70
71     // actual.forEach(jeep -> jeep.setModelPK(null));
72
73     assertThat(actual).isEqualTo(expected);
74
75 } // end of test
76

```

```
76
77 protected List<Jeep> buildExpected() {
78     List<Jeep> list = new LinkedList<>();
79     // @formatter: off
80     list.add(Jeep.builder()
81         .modelId(JeepModel.WRANGLER)
82         .trimLevel("Sport")
83         .numDoors(4)
84         .wheelSize(17)
85         .basePrice(new BigDecimal("31975.00"))
86         .build());
87     list.add(Jeep.builder()
88         .modelId(JeepModel.WRANGLER)
89         .trimLevel("Sport")
90         .numDoors(2)
91         .wheelSize(17)
92         .basePrice(new BigDecimal("28475.00"))
93         .build());
94     // @formatter: on
95
96     Collections.sort(list);
97     return list;
98 }
99 } // end of class
100
```

Screenshots of Running Application:



URL to GitHub Repository:

<https://github.com/nichspragg/Spring-Boot-Project>