


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.jeep.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);
```

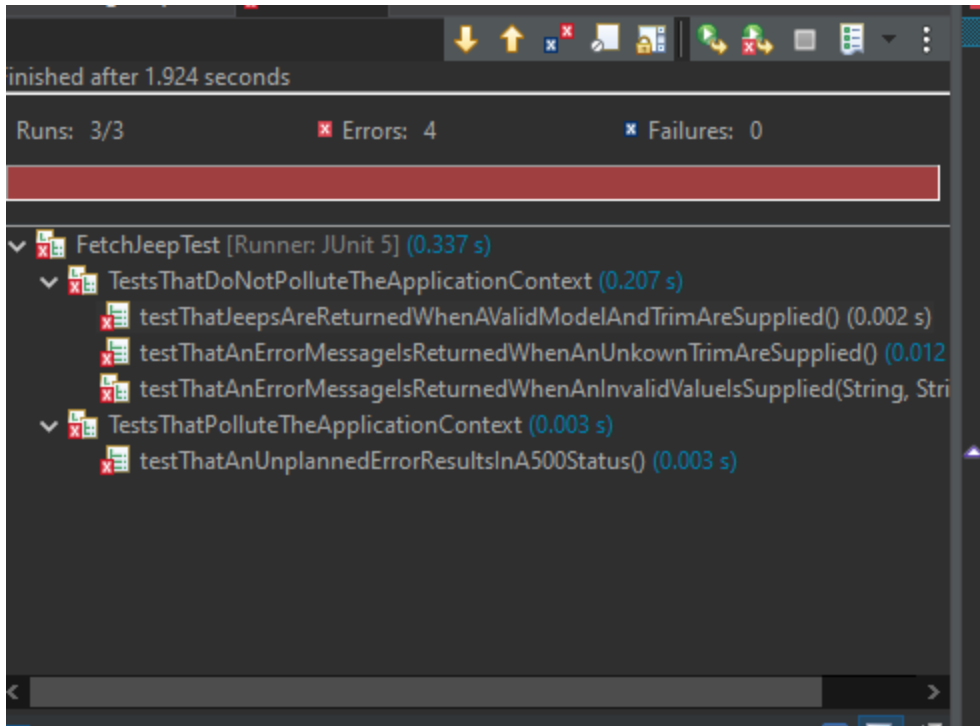
- 2) In the Jeep sales service implementation class, inject the DAO interface as an instance variable. The instance variable should be private and should be named `jeepSalesDao`. Call the DAO method from the service method and store the returned value in a local variable named `jeeps`. Return the value 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. 
 - 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. 

```

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

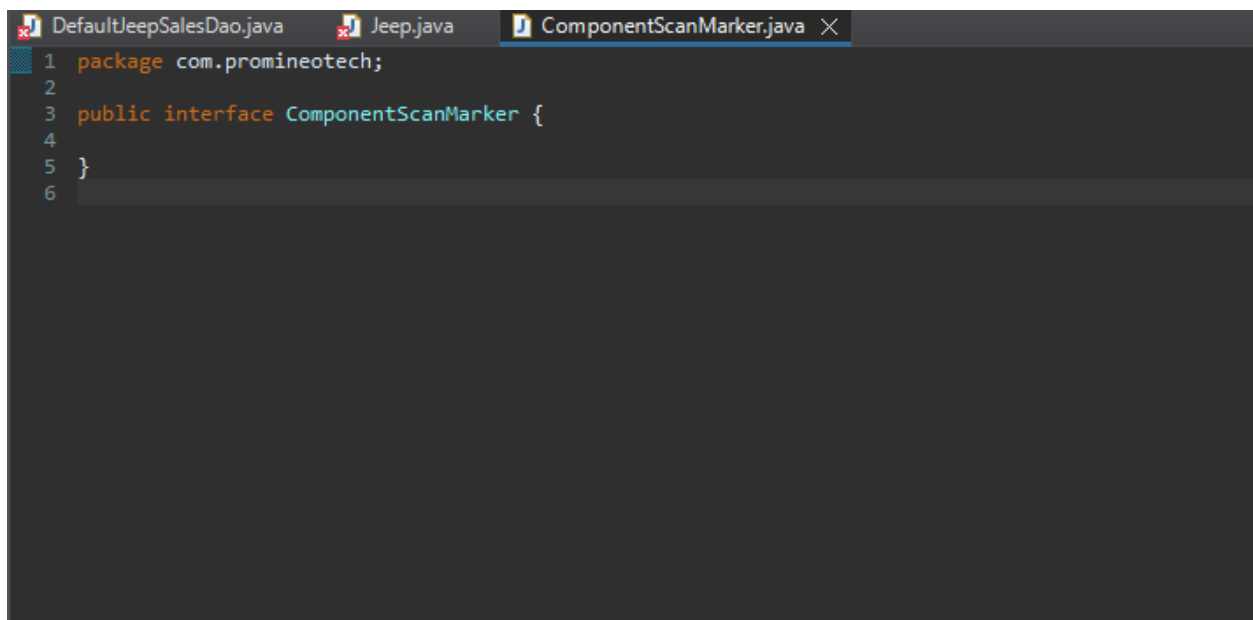
```

- 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. 



**I am having a really hard time getting all the errors fixed on this assignment. I have rewatched the videos several times since the start of spring boot and even though my code looks identical to the videos it never runs properly. I have even deleted everything and started over. Not sure if I can get some insight as to why this is happening but I will also continue to try and debug it myself.

Screenshots of Code:



```

1 package com.promineotech.jeep;
2
3 public class Constants {
4     public static final int TRIM_MAX_LENGTH = 30;
5
6     private Constants() {
7
8     }
9 }
10

```

```

1 package com.promineotech.jeep;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication(scanBasePackageClasses = { ComponentScanMarker.class })
6 public class JeepSales {
7
8     public static void main(String[] args) {
9         SpringApplication.run(JeepSales.class, args);
10    }
11 }
12

```

```

1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class BasicJeepSalesController implements JeepSalesController {
8
9     Logger log = LoggerFactory.getLogger(getClass());
10
11     @Autowired
12     private JeepSalesService jeepSalesService;
13
14     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
15         log.info("model={}, trim={}", model, trim);
16         return jeepSalesService.fetchJeeps(model, trim);
17     }
18 }
19

```

```

24
25 @Validated
26 @RequestMapping("/jeeps")
27 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
28     @Server(url = "http://localhost:8080", description = "Local server.")})
29 public interface JeepSalesController {
30
31     // @formatter:off
32     @Operation(
33         summary = "Returns a list of Jeeps",
34         description = "Returns a list of Jeeps given an optional model and/or trim",
35         responses = {
36             @ApiResponse(
37                 responseCode = "200",
38                 description = "A list of Jeeps is returned",
39                 content = @Content(mediaType = "application/json", schema = @Schema(implementation = Jeep.class))),
40             @ApiResponse(
41                 responseCode = "400",
42                 description = "The request parameters are invalid",
43                 content = @Content(mediaType = "application/json")),
44             @ApiResponse(
45                 responseCode = "404",
46                 description = "No Jeeps were found with the input criteria",
47                 content = @Content(mediaType = "application/json")),
48             @ApiResponse(
49                 responseCode = "500",
50                 description = "An unplanned error occurred",
51                 content = @Content(mediaType = "application/json"))
52         },
53         parameters = {
54             @Parameter(
55                 name = "model",
56                 allowEmptyValue = false,
57                 required = false,
58                 description = "The model name (i.e., 'WRANGLER')",
59             ),
60             @Parameter(
61                 name = "trim",
62                 allowEmptyValue = false,
63                 required = false,
64                 description = "The trim level (i.e., 'Sport')
65             )
66         }
67     )
68     @GetMapping
69     @ResponseStatus(code = HttpStatus.OK)
70     List<Jeep> fetchJeeps(
71         @RequestParam(required = false)
72         JeepModel model,
73         @Length(max = Constants.TRIM_MAX_LENGTH)
74         @Pattern(regexp = "[\\w\\s]*")
75         @RequestParam(required = false)
76         String trim);
77     // @formatter:on
78

```

```

1 package com.promineotech.jeep.dao;
2
3 import java.math.BigDecimal;
4
19
20 @Component
21 @Slf4j
22 public class DefaultJeepSalesDao implements JeepSalesDao {
23
24     Logger log = LoggerFactory.getLogger(getClass());
25
26     @Autowired
27     private NamedParameterJdbcTemplate jdbcTemplate;
28
29     @Override
30     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
31         log.debug("DAO: model={}, trim={}", model, trim);
32
33         // @formatter:off
34         String sql = ""
35             + "SELECT * "
36             + "FROM models "
37             + "WHERE model_id = :model_id AND trim_level = :trim_level";
38         // @formatter:on
39
40         Map<String, Object> params = new HashMap<>();
41         params.put("model_id", model);
42         params.put("trim_level", trim);
43
44
45         return jdbcTemplate.query(sql, params, new RowMapper<>() {
46
47             @Override
48             public Jeep mapRow(ResultSet rs, int rowNum) throws SQLException {
49                 // @formatter:off
50                 return Jeep.builder()
51                     .basePrice(new BigDecimal(rs.getString("base_price")))
52                     .modelId(JeepModel.valueOf(rs.getString("model_id")))
53                     .modelPK(rs.getLong("model_pk"))
54                     .numDoors(rs.getInt("numDoors"))
55                     .trimLevel(rs.getString("trim_level"))
56                     .wheelSize(rs.getInt("wheel_size"))
57                     .build();
58                 // @formatter:on
59             }
60         });
61     }
62 }
63

```

```

1 package com.promineotech.jeep.dao;
2
3 import java.util.List;
4
6
7 public interface JeepSalesDao {
8
9     List<Jeep> fetchJeeps(JeepModel model, String trim);
10
11 }
12

```

```

1 package com.promineotech.jee.entity;
2
3 import java.math.BigDecimal;
4 import java.util.Comparator;
5 import com.fasterxml.jackson.annotation.JsonIgnore;
6 import lombok.AccessLevel;
7 import lombok.AllArgsConstructor;
8 import lombok.Builder;
9 import lombok.Data;
10 import lombok.NoArgsConstructor;
11
12
13 @Data
14 @Builder
15 @NoArgsConstructor
16 @AllArgsConstructor
17 public class Jeep implements Comparable<Jeep>{
18     private Long modelPK;
19     private JeepModel modelId;
20     private String trimLevel;
21     private int numDoors;
22     private int wheelSize;
23     private BigDecimal basePrice;
24
25     @JsonIgnore
26     public Long getModelPK() {
27         return modelPK;
28     }
29
30     @Override
31     public int compareTo(Jeep that) {
32         // @formatter:off
33         return Comparator
34             .comparing(Jeep::getModelId)
35             .thenComparing(Jeep::getTrimLevel)
36             .thenComparing(Jeep::getNumDoors)
37             .compare(this, that);
38
39         // @formatter:on
40
41     }
42
43

```



```

20
21 @RestControllerAdvice
22 public class GlobalExceptionHandler {
23
24     Logger log = LoggerFactory.getLogger(getClass());
25
26     private enum LogStatus {
27         STACK_TRACE, MESSAGE_ONLY
28     }
29
30     @ExceptionHandler(MethodArgumentTypeMismatchException.class)
31     @ResponseStatus(code = HttpStatus.BAD_REQUEST)
32     public Map<String, Object> handleMethodArgumentTypeMismatchException(MethodArgumentTypeMismatchException e, WebRequest webRequest) {
33         return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
34     }
35
36     @ExceptionHandler(ConstraintViolationException.class)
37     @ResponseStatus(code = HttpStatus.BAD_REQUEST)
38     public Map<String, Object> handleConstraintViolationException(ConstraintViolationException e, WebRequest webRequest) {
39         return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
40     }
41     @ExceptionHandler(NoSuchElementException.class)
42     @ResponseStatus(code = HttpStatus.NOT_FOUND)
43     public Map<String, Object> handleNoSuchElementException(Exception e, WebRequest webRequest){
44         return createExceptionMessage(e, HttpStatus.NOT_FOUND, webRequest, LogStatus.MESSAGE_ONLY);
45     }
46
47     @ExceptionHandler(Exception.class)
48     @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
49     public Map<String, Object> handleException(Exception e, WebRequest webRequest) {
50         return createExceptionMessage(e, HttpStatus.INTERNAL_SERVER_ERROR, webRequest, LogStatus.STACK_TRACE);
51     }
52
53     private Map<String, Object> createExceptionMessage(Exception e, HttpStatus status, WebRequest webRequest, LogStatus logStatus) {
54         Map<String, Object> error = new HashMap<>();
55         String timestamp = ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);
56
57         if(webRequest instanceof ServletWebRequest) {
58             error.put("uri", ((ServletWebRequest) webRequest).getRequest().getRequestURI());
59         }
60
61         error.put("message", e.toString());
62         error.put("status code", status.value());
63         error.put("uri", webRequest.getContextPath());
64         error.put("timestamp", timestamp);
65         error.put("reason", status.getReasonPhrase());
66
67         if(logStatus == LogStatus.MESSAGE_ONLY) {
68             log.error("Exception: {}", e.toString());
69         }
70         else {
71             log.error("Exception: {}", e);
72         }
73     }

```

```

1 package com.promineotech.jeepp.service;
2
3 import java.util.List;
4
5
6
7 public interface JeepSalesService {
8
9     List<Jeep> fetchJeeps(JeepModel model, String trim);
10
11 }
12

```

```

1 package com.promineotech.jeeb.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 class FetchJeepTest {
6
7     @Nested
8     @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
9     @ActiveProfiles("test")
10    @Sql(scripts = {"classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
11                  "classpath:flyway/migrations/V1.1_Jeep_Data.sql"},
12        config = @SqlConfig(encoding = "utf-8"))
13    class TestsThatDoNotPolluteTheApplicationContext extends FetchJeepTestSupport {
14
15        @Autowired
16        private TestRestTemplate restTemplate;
17
18        @Test
19        void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
20            // Given: a valid model, trim and URI
21            JeepModel model = JeepModel.NRANGLER;
22            String trim = "Sport";
23            String uri = String.format("%s?model=%s&trim=%s", getBaseUrl(), model, trim);
24
25            // When: a connection is made to the URI
26            ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
27                new ParameterizedTypeReference<>() {});
28
29            // Then: a success (OK - 200) is returned
30            assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
31
32            // And: the actual list returned is the same as expected list
33            List<Jeep> actual = response.getBody();
34            List<Jeep> expected = buildExpected();
35
36            assertThat(actual).isEqualTo(expected);
37        }
38
39        @Test
40        void testThatAnErrorMessageIsReturnedWhenAnUnkownTrimAreSupplied() {
41            // Given: a valid model, trim and URI
42            JeepModel model = JeepModel.NRANGLER;
43            String trim = "Unkown value";
44            String uri = String.format("%s?model=%s&trim=%s", getBaseUrl(), model, trim);
45
46            // When: a connection is made to the URI
47            ResponseEntity<Map<String, Object>> response = restTemplate.exchange(uri, HttpMethod.GET, null,
48                new ParameterizedTypeReference<>() {});
49
50            // Then: a not found (404) status code is returned
51            assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
52
53            // And: an error message is returned
54        }
55    }
56 }

```

```

        // Then: a not found (404) status code is returned
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);

        // And: an error message is returned
        Map<String, Object> error = response.getBody();

        assertErrorMessageValid(error, HttpStatus.NOT_FOUND);
    }

    @ParameterizedTest
    @MethodSource("com.promineotech.jee.controller.FetchJeepTest#parametersForInvalidInput")
    void testThatAnErrorMessageIsReturnedWhenAnInvalidValueIsSupplied(String model, String trim, String reason) {
        // Given: a valid model, trim and URI
        String uri = String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);

        // When: a connection is made to the URI
        ResponseEntity<Map<String, Object>> response = getRestTemplate().exchange(uri, HttpMethod.GET, null,
            new ParameterizedTypeReference<>() {});

        // Then: a not found (404) status code is returned
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);

        // And: an error message is returned
        Map<String, Object> error = response.getBody();

        assertErrorMessageValid(error, HttpStatus.BAD_REQUEST);
    }

    static Stream<Arguments> parametersForInvalidInput() {
        // @formatter:off
        return Stream.of(
            arguments("WRANGLER", "@#$$^&&%", "Trim contains non-alpha-numeric chars"),
            arguments("WRANGLER", "C".repeat(Constants.TRIM_MAX_LENGTH + 1), "Trim length too long"),
            arguments("INVALID", "Sport", "Model is not enum value")
        );
    }

    protected void assertErrorMessageValid(Map<String, Object> error, HttpStatus status) {
        // @formatter:off
        assertThat(error)
            .containsKey("message")
            .containsEntry("status code", status.value())
            .containsEntry("uri", "/jeeps")
            .containsKey("timestamp")
            .containsEntry("reason", status.getReasonPhrase());
        // @formatter:on
    }
}

```

```

124     }
125 }
126 }
127 }
128
129 @Nested
130 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
131 @ActiveProfiles("test")
132 @Sql(scripts = {"classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
133               "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
134      config = @SqlConfig(encoding = "utf-8"))
135 class TestsThatPolluteTheApplicationContext extends FetchJeepTestSupport {
136     @MockBean
137     private JeepSalesService jeepSalesService;
138
139     @Test
140     void testThatAnUnplannedErrorResultsInA500Status() {
141         // Given: a valid model, trim and URI
142         JeepModel model = JeepModel.WRANGLER;
143         String trim = "Invalid";
144         String uri = String.format("%s?model=%s&trim=%s", getBaseUri(), model, trim);
145
146         doThrow(new RuntimeException("Ouch!")).when(jeepSalesService).fetchJeeps(model, trim);
147
148         // When: a connection is made to the URI
149         ResponseEntity<Map<String, Object>> response = getRestTemplate().exchange(uri, HttpMethod.GET, null,
150             new ParameterizedTypeReference<>() {});
151
152         // Then: an internal server error (500) status is returned
153         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.INTERNAL_SERVER_ERROR);
154
155         // And: an error message is returned
156         Map<String, Object> error = response.getBody();
157
158         assertErrorMessageValid(error, HttpStatus.INTERNAL_SERVER_ERROR);
159     }
160
161     protected void assertErrorMessageValid(Map<String, Object> error, HttpStatus status) {
162         // @formatter:off
163         assertThat(error)
164             .containsKey("message")
165             .containsEntry("status_code", status.value())
166             .containsEntry("uri", "/" + jeepModel)
167             .containsKey("timestamp")
168             .containsEntry("reason", status.getReasonPhrase());
169         // @formatter:on
170     }
171 }
172
173 }
174
175 }

```

Screenshots of Running Application:

```

<terminated> jeep-sales - JeepSales [Spring Boot App] C:\Users\Lally\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220903-1038\jre\bin\javaw.exe (Oct 15, 2022, 9:30:11 PM - 9:
21:30:14.359 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.cl
2022-10-15 21:30:14.696 ERROR 9148 --- [ restartedMain] o.s.b.d.LoggingFailureAnalysisReporter :

*****
APPLICATION FAILED TO START
*****

Description:

Failed to bind properties under 'logging.level.root' to org.springframework.boot.logging.LogLevel:

    Property: logging.level.root
    Value: "warn '[com.promineotech]'; debug"
    Origin: class path resource [application.yaml] - 9:11
    Reason: failed to convert java.lang.String to org.springframework.boot.logging.LogLevel (caused by java.lang.IllegalArgumentException: No enum constant org.springframework.

Action:

Update your application's configuration. The following values are valid:

    DEBUG
    ERROR
    FATAL
    INFO
    OFF
    TRACE
    WARN

```

URL to GitHub Repository:

<https://github.com/aduran92/Jeep-Sales-Week-15-Update>

