


Web API Design with Spring Boot Week 4 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:

For this week's homework you need to copy source code from the supplied resources.

For this week's homework you need to copy source code from the Source folder in the supplied resources. Wait until the instructions tell you to copy the resources or you will get errors.


1) Select some options for a Jeep order:

- Use the `data.sql` file or the jeep database tables to select options for a Jeep order. Select any one of each of the following for the order:

- i) color
 - ii) customer
 - iii) engine
 - iv) model
 - v) tire(s)
- b) Select one or more options from the options table as well. Keep in mind that some options may work better than others – but if you want to put 37-inch tires on your Jeep Renegade, so be it!
- 2) Create a new integration test class to test a Jeep order named `CreateOrderTest.java`. Create this class in `src/test/java` in the `com.promineotech.jeepp.controller` package.
- a) Add the Spring Boot Test annotations: `@SpringBootTest`, `@ActiveProfiles`, and `@Sql`. They should have the same parameters as the test created in weeks 1 and 2.
 - b) Create a test method (annotated with `@Test`) named `testCreateOrderReturnsSuccess201`.
 - c) In the test class, create a method named `createOrderBody`. This method returns a type of `String`. In this method, return a JSON object with the IDs that you picked in Step 1a and b. For example:

```
{
  "customer": "MORISON_LINA",
  "model": "WRANGLER",
  "trim": "Sport Altitude",
  "doors": 4,
  "color": "EXT_NACHO",
  "engine": "2_0_TURBO",
  "tire": "35_TOYO",
  "options": [
    "DOOR_QUAD_4",
    "EXT_AEV_LIFT",
    "EXT_WARN_WINCH",
    "EXT_WARN BUMPER_FRONT",
    "EXT_WARN BUMPER_REAR",
    "EXT_ARB_COMPRESSOR"
  ]
}
```

Make sure that the JSON is correct! If necessary, use a JSON formatter/validator like the one here: <https://jsonformatter.curiousconcept.com/>.

Produce a screenshot of the `createOrderBody()` method. 

```

67 public String createOrderBody() {
68     //@formatter: off
69     return "{\r\n"
70         + "  \"customer\": \"MORISON_LINA\", \r\n"
71         + "  \"model\": \"WRANGLER\", \r\n"
72         + "  \"trim\": \"Sport Altitude\", \r\n"
73         + "  \"doors\": 4, \r\n"
74         + "  \"color\": \"EXT_NACHO\", \r\n"
75         + "  \"engine\": \"2_0_TURBO\", \r\n"
76         + "  \"tire\": \"35_TOYO\", \r\n"
77         + "  \"options\": [\r\n"
78         + "    \"DOOR_QUAD_4\", \r\n"
79         + "    \"EXT_AEV_LIFT\", \r\n"
80         + "    \"EXT_WARN_WINCH\", \r\n"
81         + "    \"EXT_WARN BUMPER_FRONT\", \r\n"
82         + "    \"EXT_WARN BUMPER_REAR\", \r\n"
83         + "    \"EXT_ARB_COMPRESSOR\" \r\n"
84         + "  ] \r\n"
85         + "}";
86     //@formatter: on
87 }
88
89 }
90

```

Own example

```
68 public String createOrderBody() {
69
70     //@formatter: off
71     return "{\r\n"
72         + "  \"customer\": \"SCOLA_ARISTAIOS\", \r\n"
73         + "  \"model\": \"GRAND_CHEROKEE\", \r\n"
74         + "  \"trim\": \"Limited\", \r\n"
75         + "  \"doors\": 4, \r\n"
76         + "  \"color\": \"EXT_DIAMOND_BLACK\", \r\n"
77         + "  \"engine\": \"2_0_HYBRID\", \r\n"
78         + "  \"tire\": \"35_NITTO\", \r\n"
79         + "  \"options\": [\r\n"
80         + "    \"DOOR_SMITTY_FRONT\", \r\n"
81         + "    \"EXT_MOPAR_STEP_BLACK\", \r\n"
82         + "    \"EXT_DUAL_UPPER_PREMIUM\", \r\n"
83         + "    \"EXT_WARN BUMPER_FRONT\", \r\n"
84         + "    \"EXT_WARN BUMPER_REAR\", \r\n"
85         + "    \"DOOR_SMITTY_FRONT\" \r\n"
86         + "  ] \r\n"
87         + "}";
88     //@formatter: on
89 }
90
91 }
92
```

In the test method, assign the return value of the createOrderBody() method to a variable named body.

- d) In the test class, add an instance variable named serverPort to hold the port that Tomcat is listening on in the test. Annotate the variable with @LocalServerPort.
- e) Add another instance variable for an injected TestRestTemplate named restTemplate.
- f) In the test method, assign a value to a local variable named uri as follows:

```
String uri = String.format("http://localhost:%d/orders", serverPort);
```

- g) In the test method, create an HttpHeaders object and set the content type to "application/json" like this:

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
```

Make sure to import the package org.springframework.http.HttpHeaders.

- h) Create an HttpEntity object and set the request body and headers:

```
HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
```

- i) Send the request body and headers to the server. The Order class should have been copied earlier from the supplied resources. Ensure that you import `com.promineotech.jeeppromineotech.entity.Order` and not some other Order class.

```
ResponseEntity<Order> response = restTemplate.exchange(uri,  
    HttpMethod.POST, bodyEntity, Order.class);
```

- j) Add the AssertJ assertions to ensure that the response is correct. Replace the expected values to match the JSON in step 2c.

```
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);  
assertThat(response.getBody()).isNotNull();  
  
Order order = response.getBody();  
assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");  
assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);  
assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");  
assertThat(order.getModel().getNumDoors()).isEqualTo(4);  
assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");  
assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");  
assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");  
assertThat(order.getOptions()).hasSize(6);
```

- k) Produce a screenshot of the test method. 


```

30 public class CreateOrderTest {
31
32     @Autowired
33     private TestRestTemplate restTemplate;
34
35     @LocalServerPort
36     private int serverPort;
37
38     @Test
39     void testCreateOrderReturnsSuccess201() {
40         String body = createOrderBody();
41         String uri = String.format("http://localhost:%d/orders", serverPort);
42
43         HttpHeaders headers = new HttpHeaders();
44         headers.setContentType(MediaType.APPLICATION_JSON);
45
46         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
47
48         ResponseEntity<Order> response = restTemplate.exchange(uri,
49             HttpMethod.POST, bodyEntity, Order.class);
50
51         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
52         assertThat(response.getBody()).isNotNull();
53
54         Order order = response.getBody();
55         assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
56         assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
57         assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
58         assertThat(order.getModel().getNumDoors()).isEqualTo(4);
59         assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");
60         assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
61         assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
62         assertThat(order.getOptions()).hasSize(6);
63
64     }
65

```


Own example

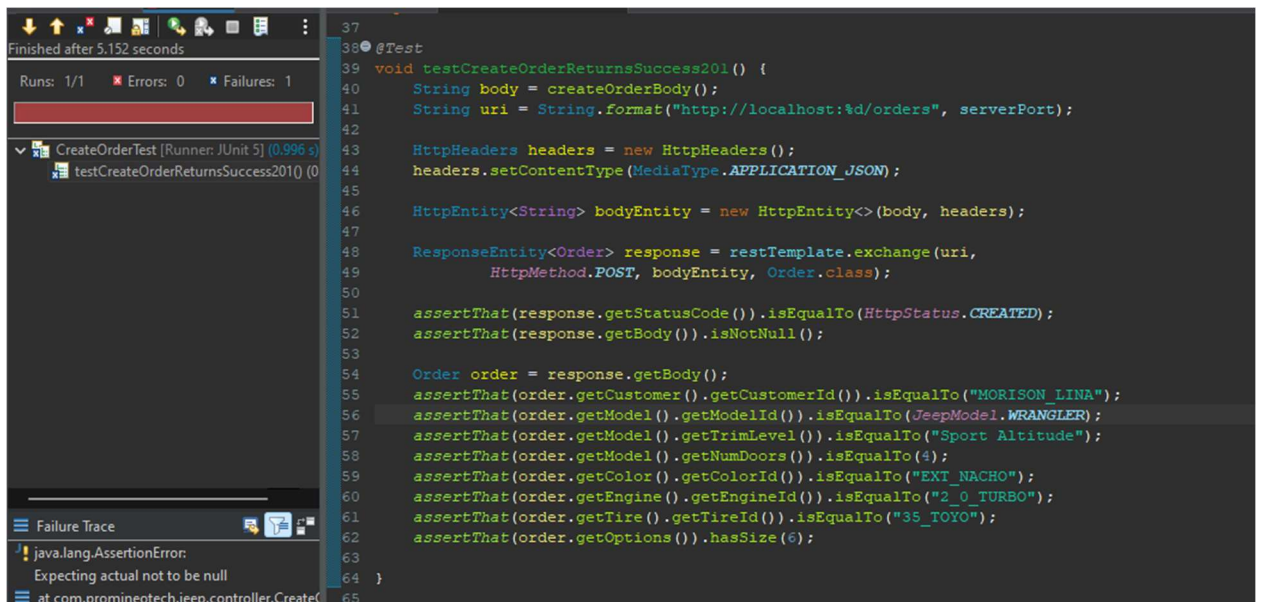
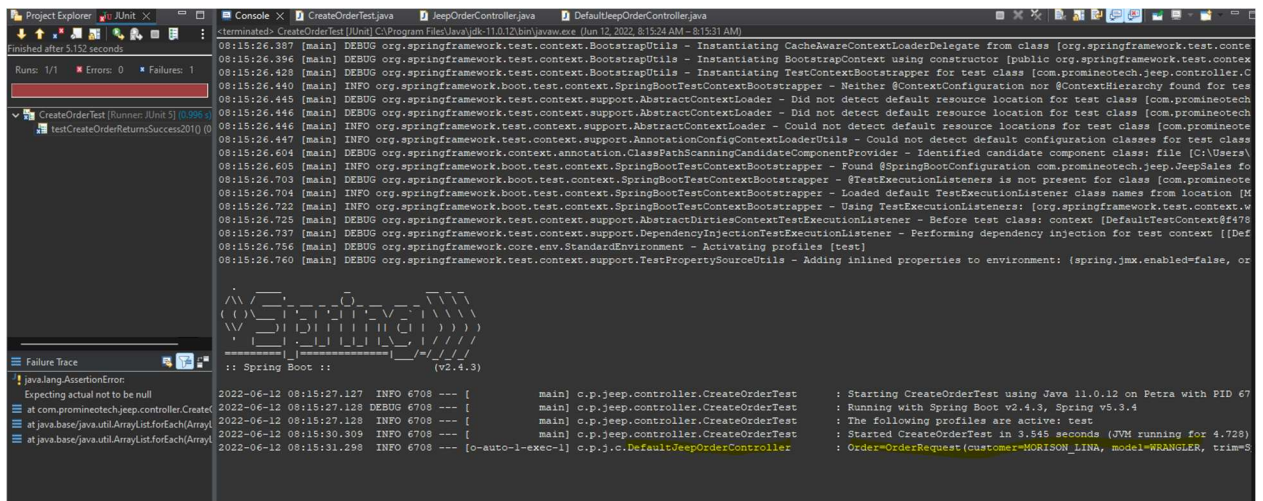
```
38 @Test
39 void testCreateOrderReturnsSuccess201() {
40     String body = createOrderBody();
41     String uri = String.format("http://localhost:%d/orders", serverPort);
42
43     HttpHeaders headers = new HttpHeaders();
44     headers.setContentType(MediaType.APPLICATION_JSON);
45
46     HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
47
48     ResponseEntity<Order> response = restTemplate.exchange(uri,
49         HttpMethod.POST, bodyEntity, Order.class);
50
51     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
52     assertThat(response.getBody()).isNotNull();
53
54
55     Order order = response.getBody();
56
57     assertThat(order.getCustomer().getCustomerId()).isEqualTo("SCOLA_ARISTAIOS");
58     assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GRAND_CHEROKEE);
59     assertThat(order.getModel().getTrimLevel()).isEqualTo("Limited");
60     assertThat(order.getModel().getNumDoors()).isEqualTo(4);
61     assertThat(order.getColor().getColorId()).isEqualTo("EXT_DIAMOND_BLACK");
62     assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_HYBRID");
63     assertThat(order.getTire().getTireId()).isEqualTo("35_NITTO");
64     assertThat(order.getOptions()).hasSize(5);
65
66 }
67
```

- 3) In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add @RequestMapping("/orders") as a class-level annotation.
 - a) Create a method in the interface to create an order (createOrder). It should return an object of type Order (see below). It should accept a single parameter of type OrderRequest as described in the video. Make sure it accepts an HTTP POST request and returns a status code of 201 (created).
 - b) Add the @RequestBody annotation to the orderRequest parameter. Make sure to add the RequestBody annotation from the org.springframework.web.bind.annotation package.
 - c) Produce a screenshot of the finished JeepOrderController interface showing no compile errors. 

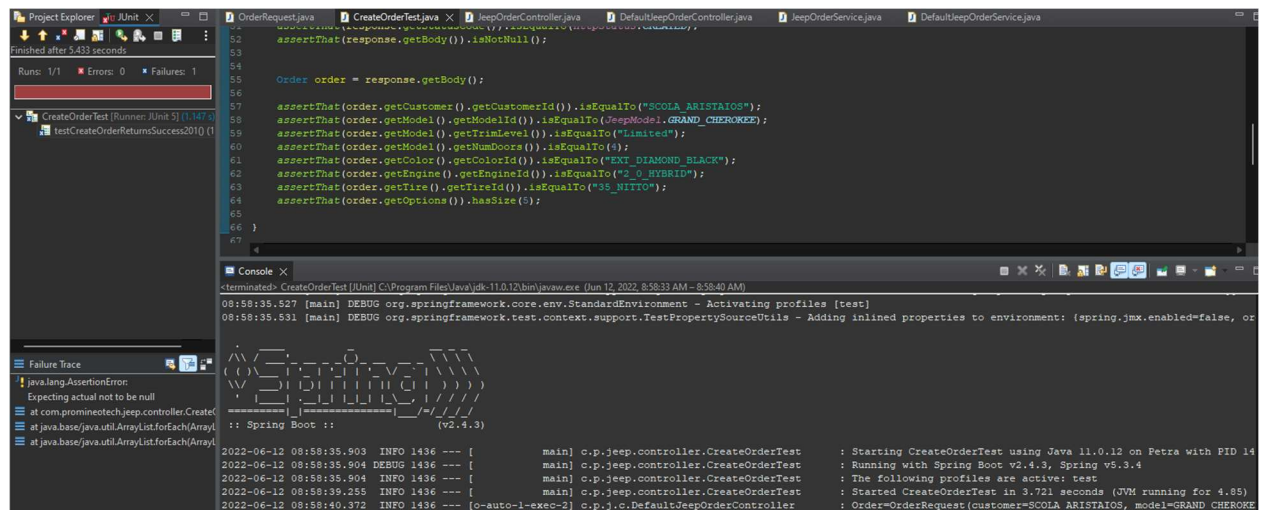

```
Console  CreateOrderTest.java  JeepOrderController.java X
25
26 @Validated
27 @RequestMapping("/orders")
28 @OpenAPIDefinition(
29     info = @Info(title = "Jeep Orders Service"),
30     servers = { @Server(url = "http://localhost:8080",description = "Local Server.")})
31 public interface JeepOrderController {
32     //@formatter:off
33     @Operation(
34         summary = "Returns an order for a Jeep",
35         description = "Returns the created Jeep",
36         responses = {
37             @ApiResponse(
38                 responseCode = "201",
39                 description = "success",
40                 content = @Content(mediaType = "application/json",
41                     schema = @Schema(implementation = Order.class))),
42             @ApiResponse(
43                 responseCode = "400",
44                 description = "bad input",
45                 content = @Content(mediaType = "application/json")),
46             @ApiResponse(
47                 responseCode = "404",
48                 description = "not found",
49                 content = @Content(mediaType = "application/json")),
50             @ApiResponse(
51                 responseCode = "500",
52                 description = "An unplanned error",
53                 content = @Content(mediaType = "application/json"))
54         },
55         parameters = {
56             @Parameter(
57                 name = "orderRequest",
58                 required = true,
59                 description = "The order as JSON"))
60     }
61 }

62
63 @PostMapping
64 @ResponseStatus(code = HttpStatus.CREATED)
65 Order createOrder(@Valid @RequestBody OrderRequest orderRequest);
66
67 }
```

- 4) Create a class that implements JeepOrderController named DefaultJeepOrderController.
 - a) Add @RestController as a class-level annotation.
 - b) Add a log line to the implementing controller method showing the input request body (orderRequest)
 - c) Run the test to show a red status bar. Produce a screenshot that shows the test method, the log line, and the red JUnit status bar. 




Own example



The screenshot shows an IDE with a project named 'jeep'. The 'Project Explorer' on the left shows the file structure. The 'JUnit' tab is active, showing a test class 'CreateOrderTest'. The 'JUnit' runner shows 'testCreateOrderReturnsSuccess2010' as the selected test. The 'Console' tab shows the execution output, including the test results and the Spring Boot version (v2.4.3). The 'Failure Trace' tab shows the error message: 'java.lang.AssertionError: Expecting actual not to be null'. The 'Test Results' tab shows the test results, including the test name, the time taken, and the status (Failed).

- 5) Find the Maven dependency `spring-boot-starter-validation` by looking it up at <https://mvnrepository.com/>. Add this repository to the project POM file (pom.xml).
- 6) Add the class-level annotation `@Validated` to the `JeepOrderController` interface.
- 7) Add Bean Validation annotations to the `OrderRequest` class as shown in the video.
 - a) Use these annotations for String types:
 - i) `@NotNull`
 - ii) `@Length(max = 30)`
 - iii) `@Pattern(regexp = "[\\w\\s]*")`
 - b) Use these annotations for integer types:
 - i) `@Positive`
 - ii) `@Min(2)`
 - iii) `@Max(4)`
 - c) Add `@NotNull` to the enum type.
 - d) Add validation to the list element (type String) by adding the validation annotations *inside* the generic definition. So, to add the String validation to the options, you would do this:

```
private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String> options;
```

Do not apply a `@NotNull` annotation to the List because if you have no options the List may be null.
 - e) Produce a screenshot of this class with the annotations. 

```

1 package com.promineotech.jee.entity;
2
3 import java.util.List;
4
5 import javax.validation.constraints.Max;
6 import javax.validation.constraints.Min;
7 import javax.validation.constraints.NotNull;
8 import javax.validation.constraints.Pattern;
9 import javax.validation.constraints.Positive;
10
11 import org.hibernate.validator.constraints.Length;
12 import lombok.Data;
13
14 @Data
15 public class OrderRequest {
16
17     @NotNull
18     @Length(max = 30)
19     @Pattern(regexp = "[\\w\\s]*")
20     private String customer;
21
22
23     @NotNull
24     private JeepModel model;
25
26     @NotNull
27     @Length(max = 30)
28     @Pattern(regexp = "[\\w\\s]*")
29     private String trim;
30
31     @Positive
32     @Min(2)
33     @Max(4)
34     private int doors;
35
36     @NotNull
37     @Length(max = 30)
38     @Pattern(regexp = "[\\w\\s]*")
39     private String color;
40
41     @NotNull
42     @Length(max = 30)
43     @Pattern(regexp = "[\\w\\s]*")
44     private String engine;
45
46     @NotNull
47     @Length(max = 30)
48     @Pattern(regexp = "[\\w\\s]*")
49     private String tire;
50
51     private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String> options;
52
53 }
54

```

- 8) In the jeep.service sub-package, create the empty (no methods yet) order service interface (named JeepOrderService) and implementation (named DefaultJeepOrderService).
 - a) Inject the interface into the order controller implementation class.
 - b) Add the @Service annotation to the service implementation class.
 - c) Create the createOrder method in the interface and implementing service. The method signature should look like this:

Order createOrder(OrderRequest orderRequest);

- d) Call the createOrder method from the controller and return the value returned by the service.
- e) Add a log line in the createOrder method and log the orderRequest parameter.
- f) Run the test CreateOrderTest again. Produce a screenshot showing that the service layer createOrder method correctly prints the log line in the console. (e.g. prints out the OrderRequest in the console from within the Service Layer).

Example from Video

```
1 package com.promineotech.jeepp.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 @RequestMapping
7 public class DefaultJeepOrderController implements JeepOrderController {
8
9     @Autowired
10    private JeepOrderService jeepOrderService;
11
12    @Override
13    public Order createOrder(OrderRequest orderRequest) {
14        log.info("Order={}", orderRequest);
15        return jeepOrderService.createOrder(orderRequest);
16    }
17
18 }
```

```
<terminated> CreateOrderTest [JUnit] C:\Program Files\Java\jdk-11.0.12\bin\java.exe (Jun 12, 2022, 8:42:01 AM - 8:42:08 AM)
:: Spring Boot :: (v2.4.3)
2022-06-12 08:42:03.459 INFO 10580 --- [main] c.p.jeepp.controller.CreateOrderTest : Starting CreateOrderTest using Java 11.0.12 on Petra with PID 1
2022-06-12 08:42:03.461 DEBUG 10580 --- [main] c.p.jeepp.controller.CreateOrderTest : Running with Spring Boot v2.4.3, Spring v5.3.4
2022-06-12 08:42:03.461 INFO 10580 --- [main] c.p.jeepp.controller.CreateOrderTest : The following profiles are active: test
2022-06-12 08:42:06.696 INFO 10580 --- [main] c.p.jeepp.controller.CreateOrderTest : Started CreateOrderTest in 3.609 seconds (JVM running for 4.865)
2022-06-12 08:42:07.997 INFO 10580 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=
```

Own Example

```
1 package com.promineotech.jeepp.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 @RequestMapping
7 public class DefaultJeepOrderController implements JeepOrderController {
8
9     @Autowired
10    private JeepOrderService jeepOrderService;
11
12    @Override
13    public Order createOrder(OrderRequest orderRequest) {
14        log.info("Order={}", orderRequest);
15        return jeepOrderService.createOrder(orderRequest);
16    }
17
18 }
```

```
<terminated> CreateOrderTest [JUnit] C:\Program Files\Java\jdk-11.0.12\bin\java.exe (Jun 12, 2022, 9:01:09 AM - 9:01:16 AM)
09:01:11.420 [main] DEBUG org.springframework.core.env.StandardEnvironment - Activating profiles [test]
09:01:11.423 [main] DEBUG org.springframework.test.context.support.TestPropertySourceUtils - Adding inlined properties to environment: (spring.jmx.enabled=false, or
:: Spring Boot :: (v2.4.3)
2022-06-12 09:01:11.783 INFO 13404 --- [main] c.p.jeepp.controller.CreateOrderTest : Starting CreateOrderTest using Java 11.0.12 on Petra with PID 1
2022-06-12 09:01:11.784 DEBUG 13404 --- [main] c.p.jeepp.controller.CreateOrderTest : Running with Spring Boot v2.4.3, Spring v5.3.4
2022-06-12 09:01:15.171 INFO 13404 --- [main] c.p.jeepp.controller.CreateOrderTest : The following profiles are active: test
2022-06-12 09:01:16.342 INFO 13404 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController : Order=OrderRequest(customer=SCOLA_ARISTAIOS, model=GRAND_CHEROK
```

- 9) In the `jeep.dao` sub-package, create the empty (no methods yet) DAO interface (named `JeepOrderDao`) and implementation (named `DefaultJeepOrderDao`).
 - a) Inject the DAO interface into the order service implementation class.
 - b) Add the `@Component` annotation to the DAO implementation class.
- 10) Replace the entire content of `JeepOrderDao.java` with the source found in `JeepOrderDao.source`. The source file is found in the Source folder of the supplied project resources.
- 11) *** The next steps require you to copy source code from the Source directory in the supplied resources. Please follow the instructions EXACTLY. Some steps require you to replace ALL the source in a file. Some steps require you to ADD source to a file.
- 12) Copy the *contents* of the file `DefaultJeepOrderDao.source` into `DefaultJeepOrderDao.java`. The source file is found in the Source folder of the supplied project resources.


In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable. Make sure you pick the import `java.util.Optional`, `java.util.List`, and `org.springframework.jdbc.core.RowMapper`.

- 13) Copy the *contents* of the file `DefaultJeepOrderService.source` into `DefaultJeepOrderService.java`. Add the source after the `createOrder()` method, but *inside* the class body. The source file is found in the Source folder of the supplied project resources.
- In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable.

- 14) In `DefaultJeepOrderService.java`, work with the method `createOrder`.
 - a) Add the `@Transactional` annotation to the `createOrder` method.
 - b) In the `createOrder` method call the copied methods: `getCustomer`, `getModel`, `getColor`, `getEngine`, `getTire` and `getOption`, assigning the return values of these methods to variables of the appropriate types.
 - c) Calculate the price, including all options.

- 15) In `JeepOrderDao.java` and `DefaultJeepOrderDao.java`, add the method:

```
Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire
tire, BigDecimal price, List<Option> options);
```

- a) Call the `jeepOrder.Dao.saveOrder` method from the `jeepOrderSalesService.createOrder` service. Produce a screenshot of the `jeepOrderSalesService.createOrder` method. 


```

23 @Service
24 public class DefaultJeepOrderService implements JeepOrderService {
25
26     @Autowired
27     private JeepOrderDao jeepOrderDao;
28
29     @Transactional
30     @Override
31     public Order createOrder(OrderRequest orderRequest) {
32
33         Customer customer = getCustomer(orderRequest);
34         Jeep jeep = getModel(orderRequest);
35         Color color = getColor(orderRequest);
36         Engine engine = getEngine(orderRequest);
37         Tire tire = getTire(orderRequest);
38         List<Option> options = getOption(orderRequest);
39
40         BigDecimal price = jeep.getBasePrice().add(color.getPrice()).add(engine.getPrice()).add(tire.getPrice());
41
42         for(Option option : options) {
43             price.add(option.getPrice());
44         }
45         return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
46     }
47

```

b) Write the implementation of the saveOrder method in the DAO.

- i) Call the supplied generateInsertSql method, passing in the customer, jeep, color, engine, tire and price. Assign the return value of the method to a SqlParameter object.
- ii) Call the update method on the NamedParameterJdbcTemplate object, passing in a KeyHolder object as shown in the video. Create the KeyHolder like this:


```
KeyHolder keyHolder = new GeneratedKeyHolder();
```

Be sure to extract the order primary key from the KeyHolder object into a variable of type Long named orderPK.

- iii) Write a method named saveOptions as shown in the video. This method should have the following method signature:

```
private void saveOptions(List<Option> options, Long orderPK)
```

For each option in the Options list, call the supplied generateInsertSql method passing the parameters option and order primary key (orderPK). Call the update method on the NamedParameterJdbcTemplate object.

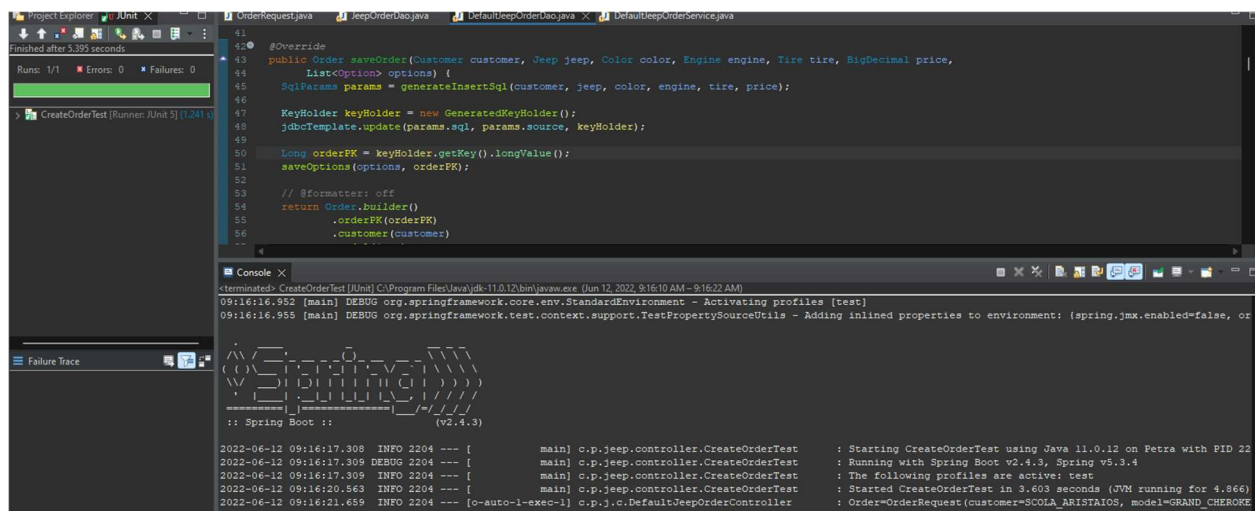
- iv) In the saveOrder method in the DAO implementation, return an Order object using the Order.builder. The Order should include orderPK, customer, jeep (model), color, engine, tire, options and price.
- v) Produce a screenshot of the saveOrder method. 

```

42 @Override
43 public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price,
44     List<Option> options) {
45     SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
46
47     KeyHolder keyHolder = new GeneratedKeyHolder();
48     jdbcTemplate.update(params.sql, params.source, keyHolder);
49
50     Long orderPK = keyHolder.getKey().longValue();
51     saveOptions(options, orderPK);
52
53     // @formatter: off
54     return Order.builder()
55         .orderPK(orderPK)
56         .customer(customer)
57         .model(jeep)
58         .color(color)
59         .engine(engine)
60         .tire(tire)
61         .options(options)
62         .price(price)
63         .build();
64     // @formatter: on
65 }
66
67 private void saveOptions(List<Option> options, Long orderPK) {
68     for(Option option : options) {
69         SqlParams params = generateInsertSql(option, orderPK);
70         jdbcTemplate.update(params.sql, params.source);
71     }

```

- c) Run the integration test in CreateOrderTest. Produce a screenshot of the test method that shows the green JUnit status bar, the console output, and the test class. 🖥️



URL to GitHub Repository:

<https://github.com/ailimutan/alect-springboot-week16-assignment>