

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

Coding Steps:

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:
 - ✓ a) 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.jeep.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:

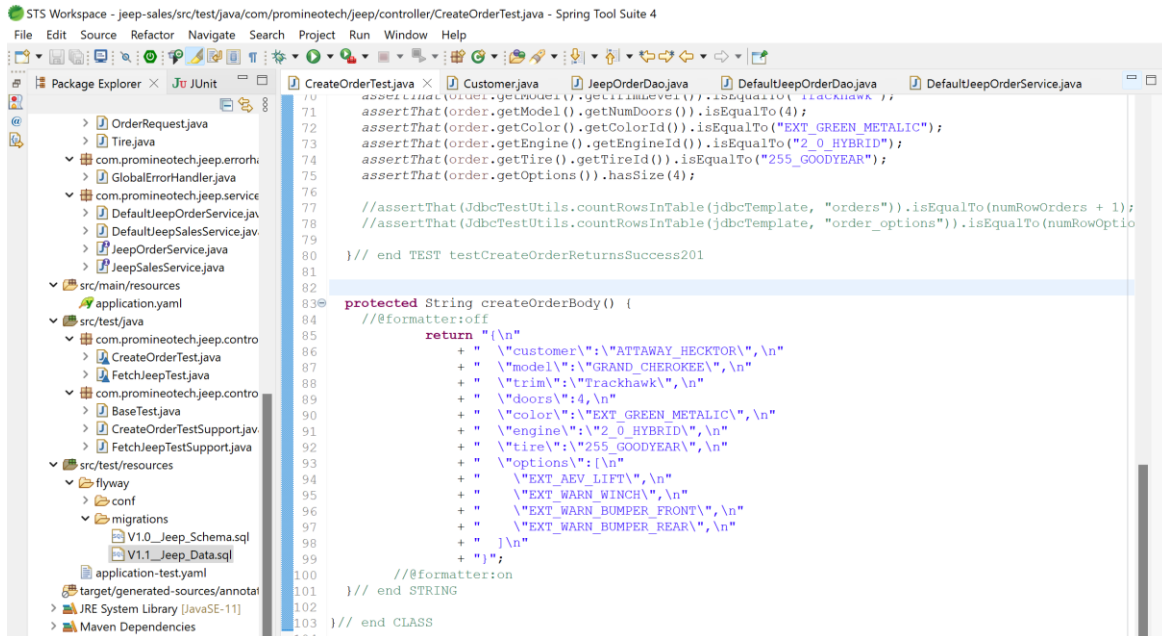
```
{ "customer": "MORISON_LINA",  
  "model": "WRANGLER",  
  "trim": "Sport Altitude",  
  "doors": 4,  
  "color": "EXT_NACHO",  
  "engine": "2_0_TURBO",  
  "tire": "35_TOYO",  
  "options": [  
    "DOOR_QUAD_4",  
    "EXT_AEV_LIFT",  
    "EXT_WARN_WINCH",  
    "EXT_WARN BUMPER_FRONT",  
  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

```

    "EXT_ARB_COMPRESSOR"
  }
}

```

✓ Produce a screenshot of the createOrderBody() method. 



```

71  assertThat(order.getModel().getNumDoors()).isEqualTo(4);
72  assertThat(order.getColor().getColorId()).isEqualTo("EXT_GREEN_METALIC");
73  assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_HYBRID");
74  assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
75  assertThat(order.getOptions()).hasSize(4);
76
77  //assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders")).isEqualTo(numRowOrders + 1);
78  //assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options")).isEqualTo(numRowOptions + 1);
79
80  // end TEST testCreateOrderReturnsSuccess201
81
82
83  protected String createOrderBody() {
84      // @formatter:off
85      return "{\n"
86          + "  \"customer\": \"ATTAWAY_HECKTOR\",\n"
87          + "  \"model\": \"GRAND_CHEROKEE\",\n"
88          + "  \"trim\": \"Trackhawk\",\n"
89          + "  \"doors\": 4,\n"
90          + "  \"color\": \"EXT_GREEN_METALIC\",\n"
91          + "  \"engine\": \"2_0_HYBRID\",\n"
92          + "  \"tire\": \"255_GOODYEAR\",\n"
93          + "  \"options\": [\n"
94          + "    \"EXT_AEV_LIFT\",\n"
95          + "    \"EXT_WARN_WINCH\",\n"
96          + "    \"EXT_WARN BUMPER_FRONT\",\n"
97          + "    \"EXT_WARN BUMPER_REAR\",\n"
98          + "  ]\n"
99          + "}";
100      // @formatter:on
101  } // end STRING
102
103  } // end CLASS

```

✓ 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.jee.entity.Order and not some other Order class.

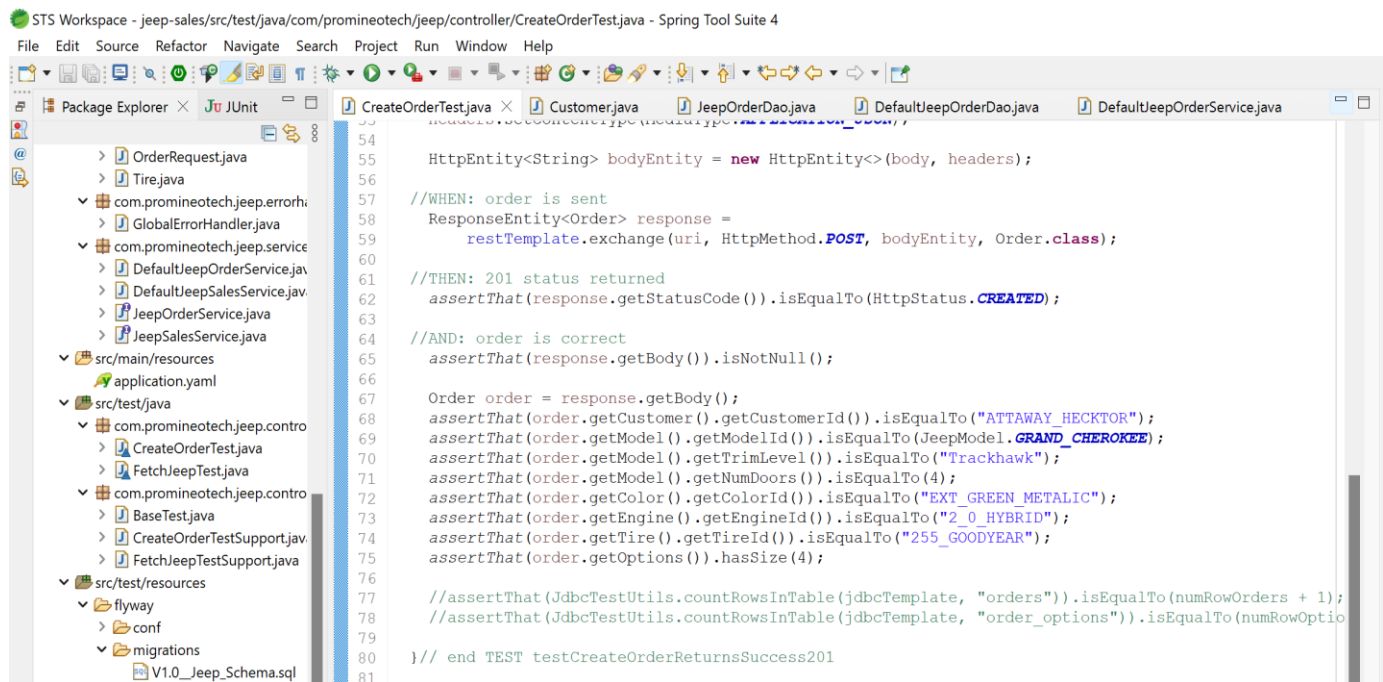
```
ResponseEntity<Order> response = restTemplate.exchange(uri,
    HttpMethod.POST, bodyEntity, Order.class);
```

- ✓ 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. 




```
STS Workspace - jeep-sales/src/test/java/com/promineotech/jeep/controller/CreateOrderTest.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit CreateOrderTest.java Customer.java JeepOrderDao.java DefaultJeepOrderDao.java DefaultJeepOrderService.java
OrderRequest.java Tire.java
com.promineotech.jee.errorh
GlobalErrorHandler.java
com.promineotech.jee.service
DefaultJeepOrderService.java
DefaultJeepSalesService.java
JeepOrderService.java
JeepSalesService.java
src/main/resources
application.yaml
src/test/java
com.promineotech.jee.contro
CreateOrderTest.java
FetchJeepTest.java
com.promineotech.jee.contro
BaseTest.java
CreateOrderTestSupport.java
FetchJeepTestSupport.java
src/test/resources
flyway
conf
migrations
V1.0_Jeep_Schema.sql
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
//WHEN: order is sent
ResponseEntity<Order> response =
    restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, Order.class);
//THEN: 201 status returned
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
//AND: order is correct
assertThat(response.getBody()).isNotNull();
Order order = response.getBody();
assertThat(order.getCustomer().getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");
assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GRAND_CHEROKEE);
assertThat(order.getModel().getTrimLevel()).isEqualTo("Trackhawk");
assertThat(order.getModel().getNumDoors()).isEqualTo(4);
assertThat(order.getColor().getColorId()).isEqualTo("EXT_GREEN_METALIC");
assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_HYBRID");
assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
assertThat(order.getOptions()).hasSize(4);
//assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders")).isEqualTo(numRowOrders + 1);
//assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options")).isEqualTo(numRowOptio
} // end TEST testCreateOrderReturnsSuccess201
```

- ✓ 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 errors. 

```

1 package com.promineotech.jeepp.controller;
2
3 import javax.validation.Valid;
4 import java.util.List;
5
6 import org.springframework.http.HttpStatus;
7 import org.springframework.validation.annotation.Validated;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RequestParam;
12 import org.springframework.web.bind.annotation.ResponseStatus;
13
14 import com.promineotech.jeepp.Constants;
15 import com.promineotech.jeepp.entity.Jeep;
16 import com.promineotech.jeepp.entity.JeepModel;
17 import com.promineotech.jeepp.entity.Order;
18 import com.promineotech.jeepp.entity.OrderRequest;
19 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
20 import io.swagger.v3.oas.annotations.Operation;
21 import io.swagger.v3.oas.annotations.Parameter;
22 import io.swagger.v3.oas.annotations.info.Info;
23 import io.swagger.v3.oas.annotations.media.Content;
24 import io.swagger.v3.oas.annotations.media.Schema;
25 import io.swagger.v3.oas.annotations.parameters.RequestBody;
26 import io.swagger.v3.oas.annotations.responses.ApiResponse;
27 import io.swagger.v3.oas.annotations.servers.Server;
28
29 @Validated
30 @RequestMapping("/orders")
31 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"),
32 servers = {@Server(url = "http://localhost:8080", description = "Local server.")})
33
34 public interface JeepOrderController {
35
36     /**@formatter:off
37     @Operation(
38         summary = "Create an order for a Jeep",
39         description = "Returns the created Jeep",
40         responses = {
41             @ApiResponse(
42                 responseCode = "201",
43                 description = "The created Jeep is returned.",
44                 content = @Content(
45                     mediaType = "application/json",
46                     schema = @Schema(implementation = Order.class)),
47             @ApiResponse(
48                 responseCode = "400",
49                 description = "The request parameters are invalid.",
50                 content = @Content(mediaType = "application/json")),
51             @ApiResponse(
52                 responseCode = "404",
53                 description = "A Jeep component was not found with input criteria.",
54                 content = @Content(mediaType = "application/json")),
55             @ApiResponse(
56                 responseCode = "500",
57                 description = "An unplanned error occurred.",
58                 content = @Content(mediaType = "application/json"))
59         },//end Responses
60         parameters = {
61             @Parameter(
62                 name = "orderRequest",
63                 required = true,
64                 description = "The order as JSON")
65         },//end parameters
66     )//end Operation Annotation
67     /**@formatter:on
68
69     @PostMapping
70     @ResponseStatus(code = HttpStatus.CREATED)
71     Order createOrder(@Valid @RequestBody OrderRequest orderRequest);
72
73     |
74
75
76 }// end INTERFACE JeepSalesController
77

```

- ✓ 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. 

I am too far past this point and have finished the coding in the videos. I no longer have a red status bar

- ✓ 5) Find the Maven dependency spring-boot-starter-validation. Add this to the project POM file.

- ✓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)`

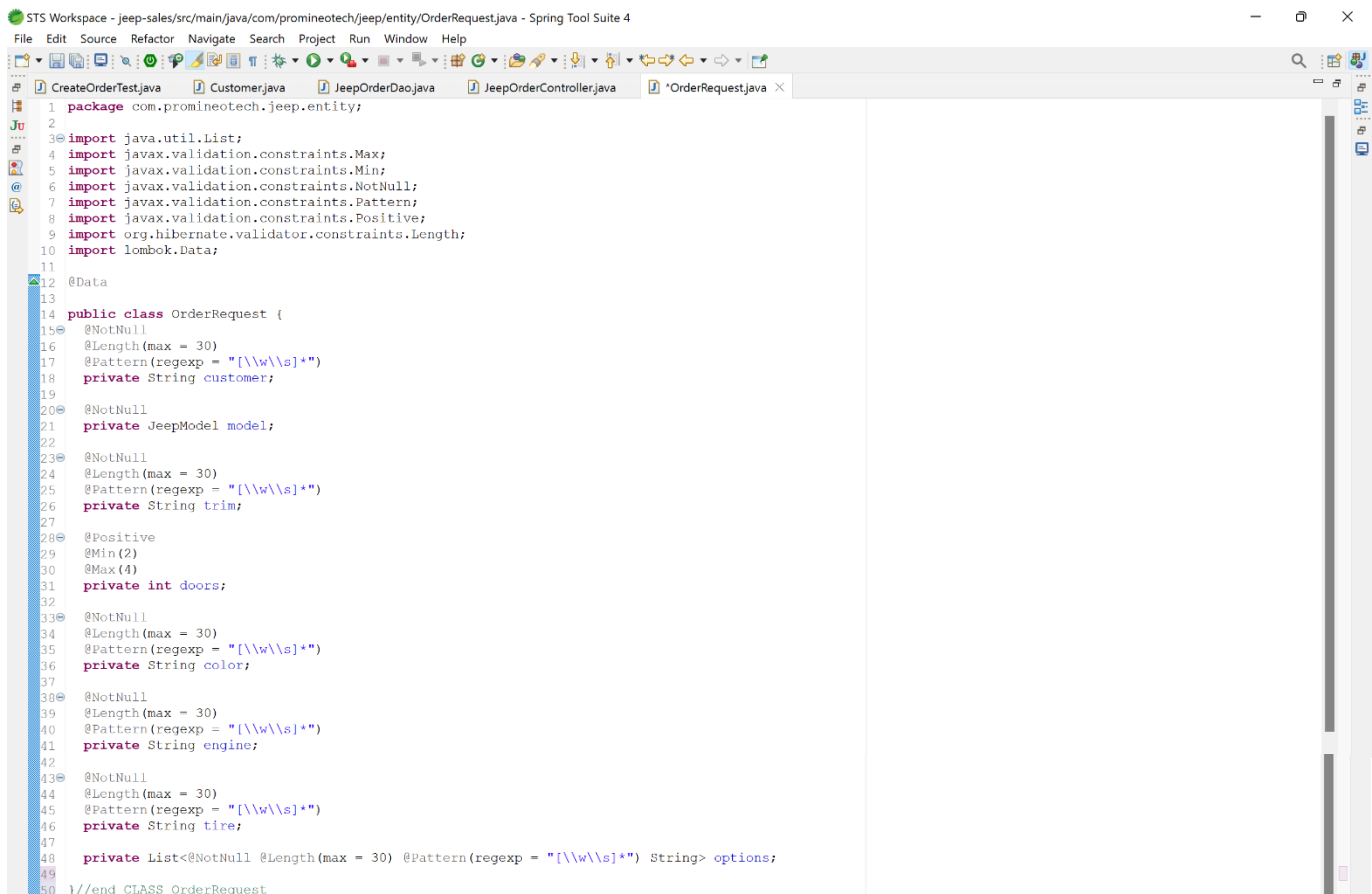
✓c) Add `@NotNull` to the enum type.

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

- ✓ 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);
```

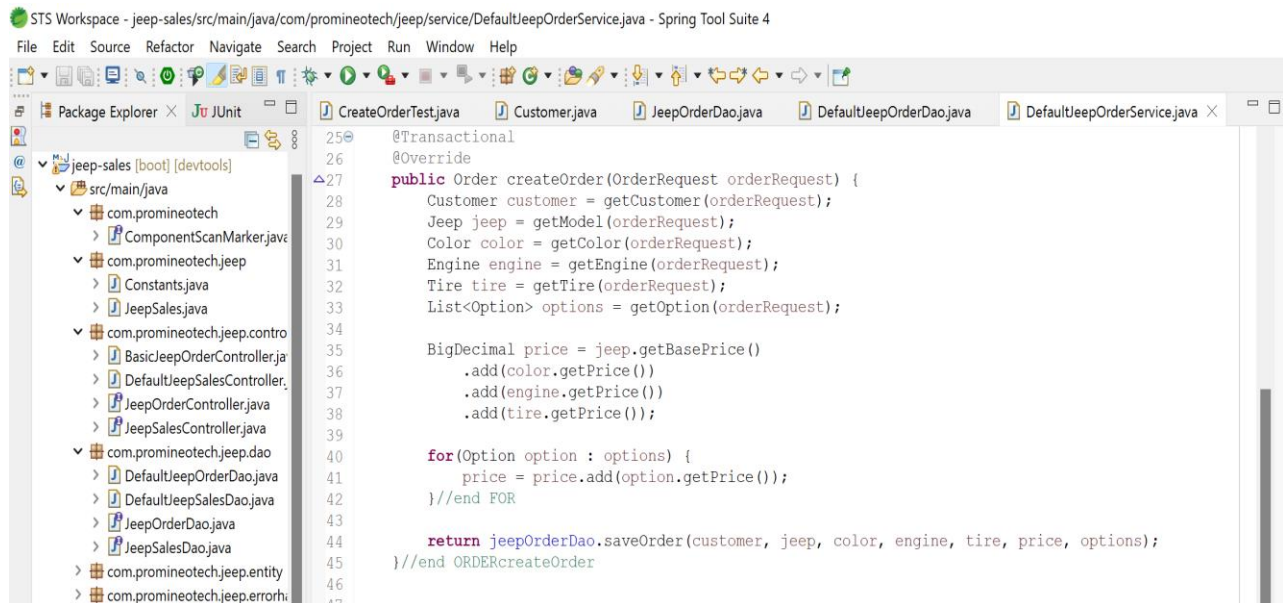
- ✓ d) Call the `createOrder` method from the controller and return the value returned by the service.
- ✓ e) Add a log line in the `createOrder` method and log the `orderRequest` parameter.
- ✓ f) Run the test `CreateOrderTest` again. Produce a screenshot showing that the service layer `createOrder`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

I am too far past this point and have finished the coding in the videos.

- ✓ 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.
- ✓ 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.
- ✓ 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. 



```

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


```

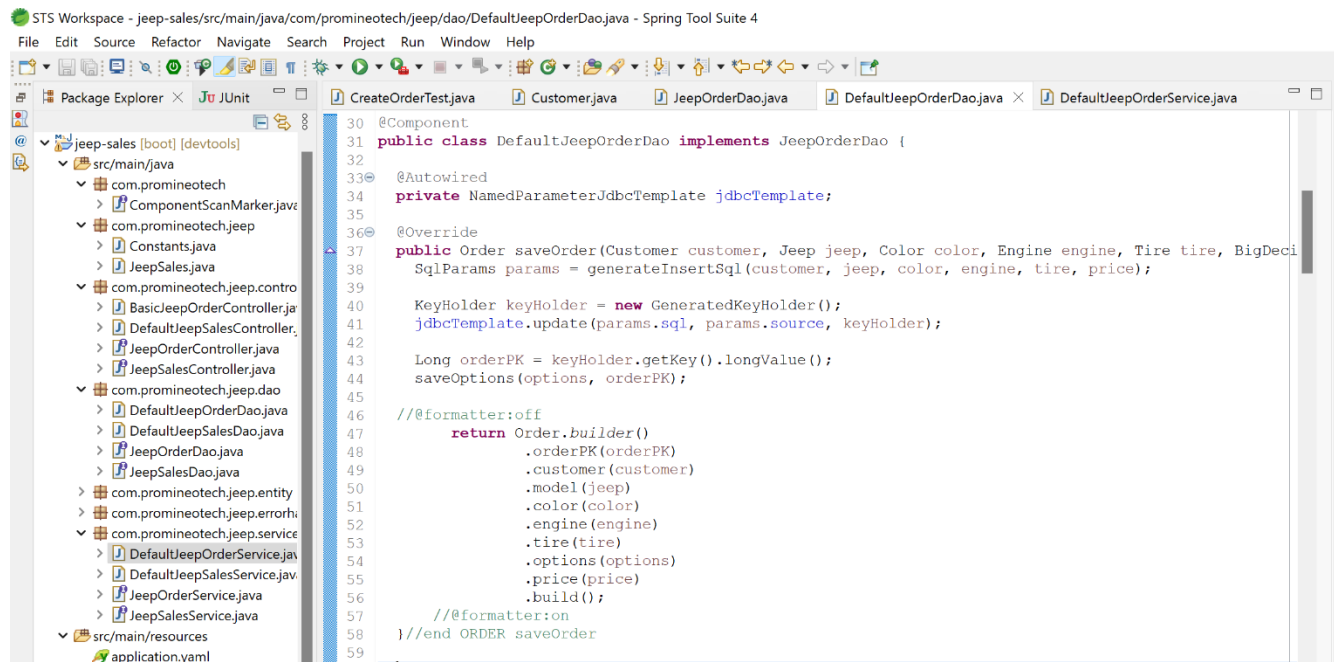
- ✓ b) Write the implementation of the `saveOrder` method in the DAO.
- ✓ i) Call the supplied `generateInsertSql` method, passing in the customer, jeep, color, engine, tire and price. Assign the return value of the method to a `SqlParams` object.
 - ✓ ii) Call the `update` method on the `NamedParameterJdbcTemplate`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


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

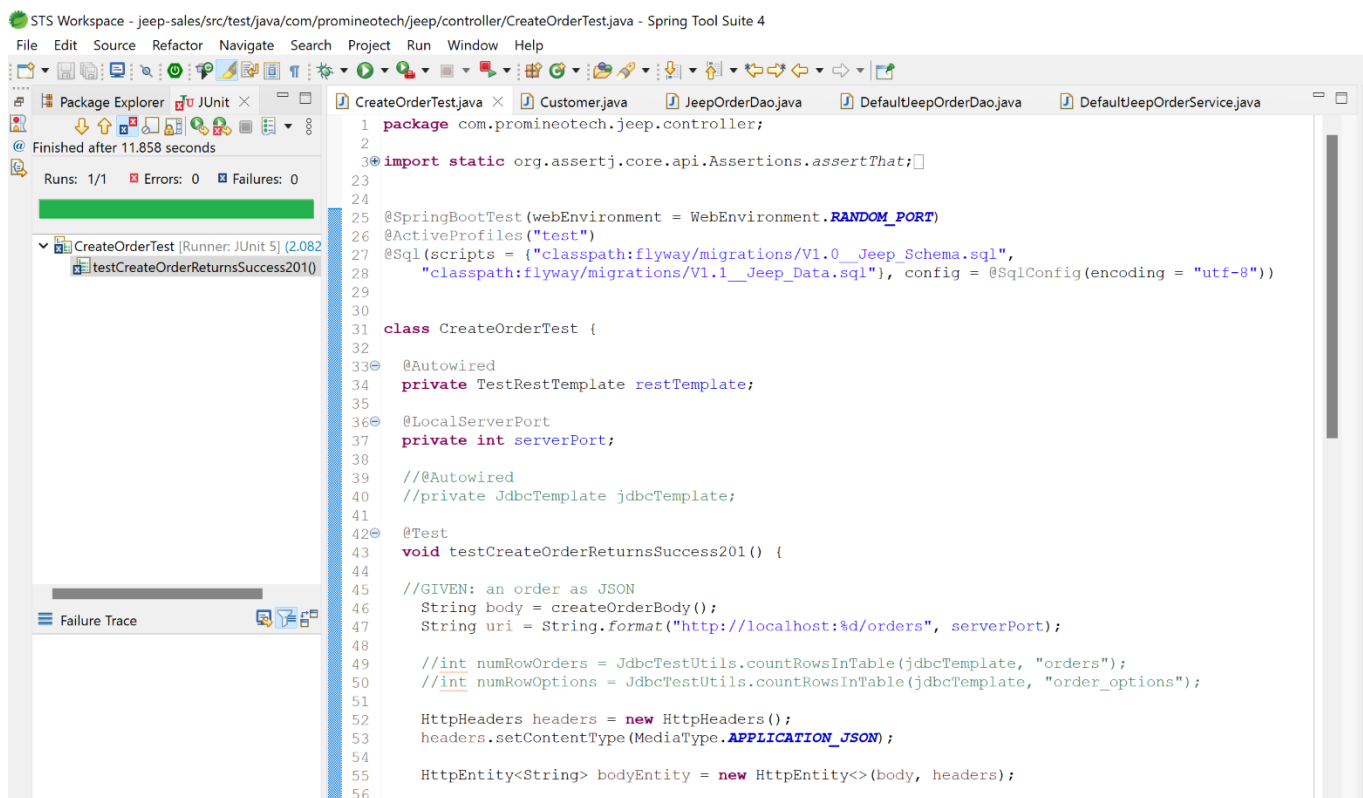
Be sure to extract the order primary key from the `KeyHolder` object into a variable of type `Long` named `orderPK`.
 - ✓ iii) Write a method named `saveOptions` as shown in the video. This method should have the following method signature:


```
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. 



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/JaxYoungblood/Week16-SpringBootCodingAssignment.git>