

# 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.

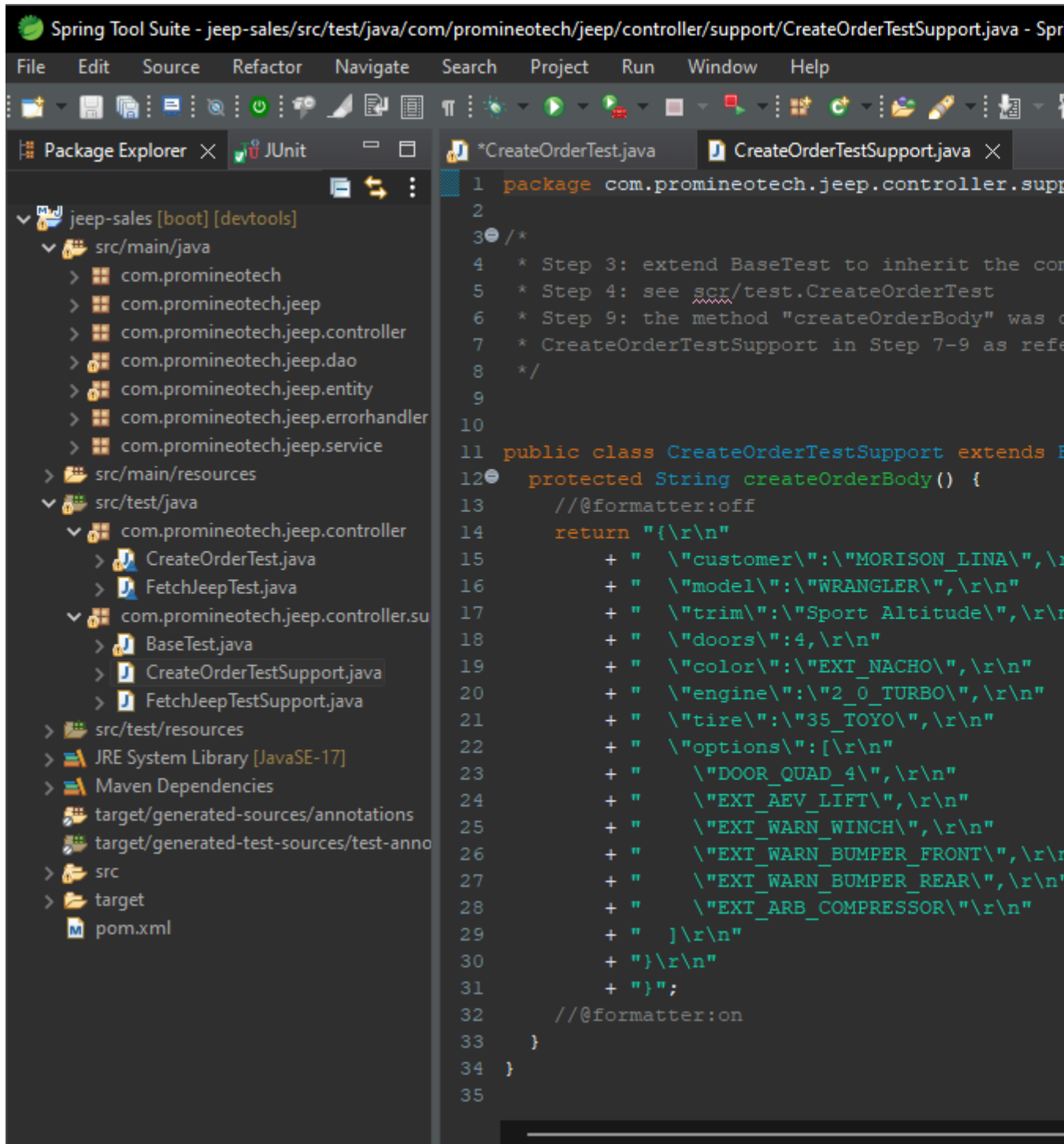
- 1) Select some options for a Jeep order:
  - a) Use the `data.sql` file or the jeep database tables to select options for a Jeep order. Select any one of each of the following for the order:

- 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.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. 



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.jeepprom.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. 

Finished after 4.72 seconds


Runs: 1/1 x Errors: 0 x Failures: 1

▼ CreateOrderTest [Runner: JUnit 5] (0.727 s)  
x testCreateOrderReturnsSuccess201() (0.727 s)

## Failure Trace


! java.lang.AssertionError:  
Expecting actual not to be null  
at com.promineotech.jeep.controller.CreateOrderTest.testCreateOrd  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

```
1 package com.promineotech.jeep.controller;  
2  
3 import static org.assertj.core.api.Assertions.*;  
4 import org.junit.jupiter.api.Test;  
5 import org.springframework.boot.test.context.SpringBootTest;  
6 import org.springframework.boot.test.context.SpringBootTest;  
7 import org.springframework.http.ResponseEntity;  
8 import org.springframework.http.HttpHeaders;  
9 import org.springframework.http.HttpMethod;  
10 import org.springframework.http.HttpStatus;  
11 import org.springframework.http.MediaType;  
12 import org.springframework.http.ResponseEntity;  
13 import org.springframework.test.context.ActiveProfiles;  
14 import org.springframework.test.context.jdbc.Sql;  
15 import org.springframework.test.context.jdbc.SqlConfig;  
16 import com.promineotech.jeep.controller.support.CreateOrderSupport;  
17 import com.promineotech.jeep.entity.JeepModel;  
18 import com.promineotech.jeep.entity.Order;  
19  
20 * Objective: Behavior Driven Development for CRED  
21  
22  
23  
24  
25  
26 @SpringBootTest(webEnvironment = WebEnvironment.R  
27 /*  
28 * @SpringBootTest with webEnvironment creates a l  
29 */  
30 @ActiveProfiles("test")  
31 /*  
32 * @ActiveProfiles will load up the H2 database  
33 */  
34 @Sql(scripts = {"classpath:flyway/migrations/V1.0  
35 "classpath:flyway/migrations/V1.1__Jeep_Data  
36 config = @SqlConfig(encoding = "utf-8")  
37 class CreateOrderTest extends CreateOrderTestSupp  
38  
39 /*  
40 * we are going to create some JSON to throw at t  
41 */  
42 @Test  
43 void testCreateOrderReturnsSuccess201() {  
44 // Given: an order as JSON  
45 String body = createOrderBody();  
46 String uri = getBaseUrlForOrders();  
47 HttpHeaders headers = new HttpHeaders();  
48 headers.setContentType(MediaType.APPLICATION_  
49 ResponseEntity<String> bodyEntity = new HttpEntit  
50  
51 // When: the order is sent - Create a JeepOrd  
52 ResponseEntity<Order> response = getRestTempl  
53  
54 // Then: a 201 status is returned  
55 assertThat(response.getStatusCode()).isEqualTo  
56  
57 // And: the returned order is correct  
58 assertThat(response.getBody()).isNotNull();  
59 Order order = response.getBody();  
60  
61 assertThat(order.getCustomer().getCustomerId()).isE  
62 assertThat(order.getModel().getModelId()).isE  
63 assertThat(order.getModel().getTrimLevel()).isE  
64 assertThat(order.getModel().getNumDoors()).isE  
65 assertThat(order.getColor().getColorId()).isE  
66 assertThat(order.getEngine().getEngineId()).isE  
67 assertThat(order.getTire().getTireId()).isE  
68 assertThat(order.getOptions().hasSize(6);  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115
```

- 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. 

```
<terminated> CreateOrderTest [JUnit] C:\Users\darryl.nichols\Desktop\Back End Course Tools\Spring Tool Suite\sts-4.15.1.RELEASE\plu
```

```
2022-08-19 16:29:51.945 INFO 11584 --- [main] c.p.jee.p.controller.CreateOrderTest : Starting CreateOrderTest using Java 17.0.
2022-08-19 16:29:51.946 DEBUG 11584 --- [main] c.p.jee.p.controller.CreateOrderTest : Running with Spring Boot v2.7.2, Spring v
2022-08-19 16:29:51.947 INFO 11584 --- [main] c.p.jee.p.controller.CreateOrderTest : The following 1 profile is active: "test"
2022-08-19 16:29:54.820 INFO 11584 --- [main] c.p.jee.p.controller.CreateOrderTest : Started CreateOrderTest in 3.14 seconds
```

- 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 named it “BasicJeepOrderController”



File Edit Source Refactor Navigate Search Project Run Window Help



Package Explorer JUnit X



Finished after 4.448 seconds

Runs: 1/1 Errors: 0 Failures: 1



▼ CreateOrderTest [Runner: JUnit 5] (0.761 s)  
    x testCreateOrderReturnsSuccess2010 (0.761 s)

Failure Trace

CreateOrderTest.java JeepOrderController.java BasicJeepOrderController.java

```
1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13 /*
14  * Step 22: implement the createOrder method a
15  * Step 23: add @RestController, and log the l
16  * with Slf4j which is from Lombok
17  * Step 24: go to src/test/jeep.controller.Cre
18  * Step 28: Create an JeepOrderService interfa
19  * Step 29: code createOrder to return createO
20  */
21
22
23
24
25
26 @RestController
27 @Slf4j
28 public class BasicJeepOrderController implemen
29
30 @Autowired
31 private JeepOrderService jeepOrderService;
32
33 @Override
34 public Order createOrder(OrderRequest orderR
35     log.debug("Order={}", orderRequest);
36     return jeepOrderService.createOrder(orderR
37 }
38
39 }
40
```

Problems Javadoc Declaration Console X

&lt;terminated&gt; CreateOrderTest [JUnit] C:\Users\darryl.nichols\Desktop\Back

:: Spring Boot :: (v2.7.2)

2022-08-22 09:11:56.213 INFO 33244 --- [

2022-08-22 09:11:56.214 DEBUG 33244 --- [


2022-08-22 09:11:56.215 INFO 33244 --- [

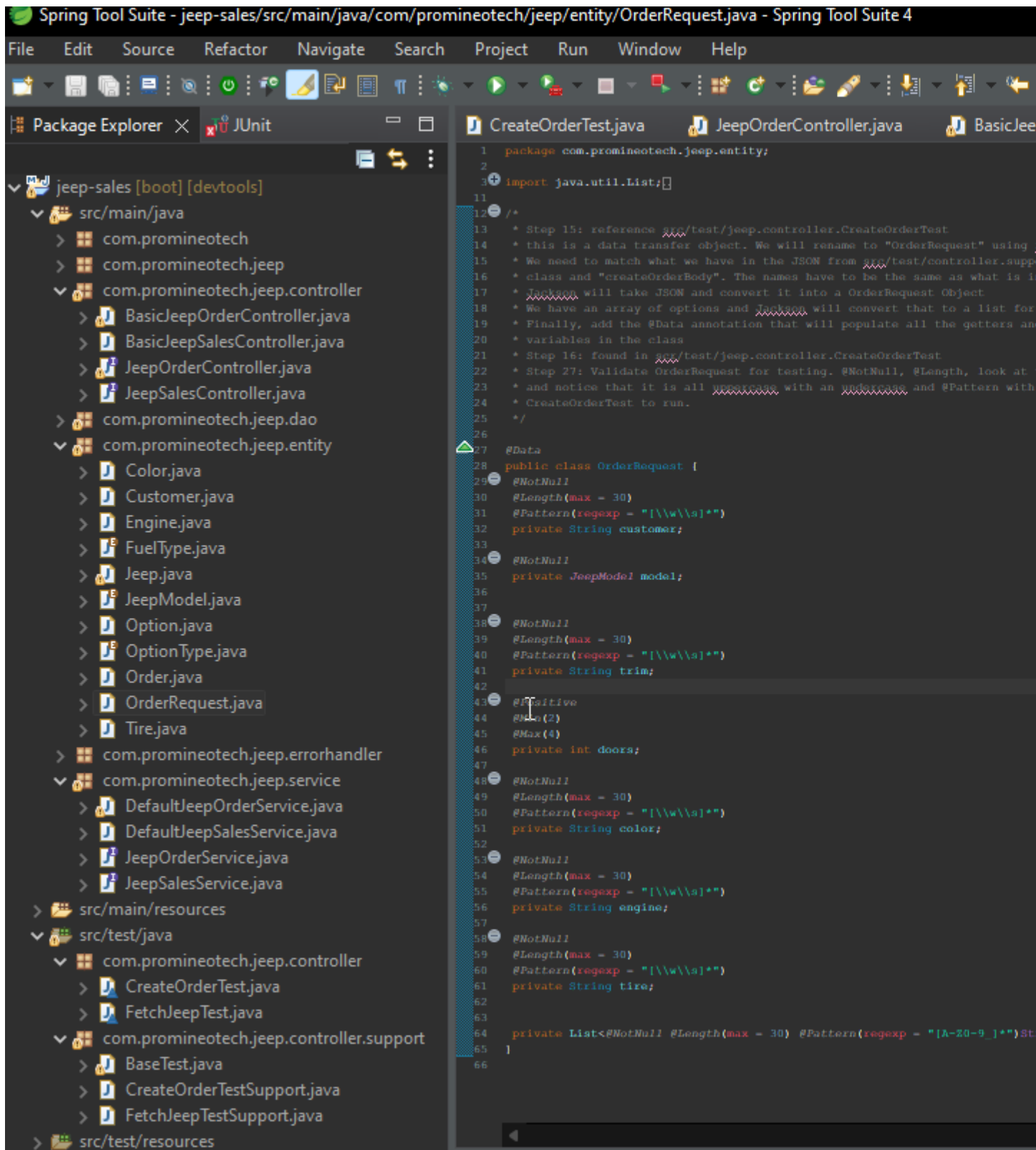
2022-08-22 09:11:59.209 INFO 33244 --- [

2022-08-22 09:11:59.210 DEBUG 33244 --- [

- 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. 



- 8) In the `jeep.service` sub-package, create the empty (no methods yet) order service interface (named `JeepOrderService`) and implementation (named `DefaultJeepOrderService`).



- 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. 

```
DefaultJeepOrderService.java X CreateOrderTest.java DefaultJeepOrderDao.java JeepOrderDao.java
24  * Step 33: Create the dao implementing class
25  */
26
27  @Service
28  public class DefaultJeepOrderService implements JeepOrderService {
29
30      @Autowired
31      private JeepOrderDao jeepOrderDao;
32
33      @Transactional
34      @Override
35      public Order createOrder(OrderRequest orderRequest) {
36          Customer customer = getCustomer(orderRequest);
37          Jeep jeep = getModel(orderRequest);
38          Color color = getColor(orderRequest);
39          Engine engine = getEngine(orderRequest);
40          Tire tire = getTire(orderRequest);
41          List<Option> options = getOption(orderRequest);
42
43          BigDecimal price = jeep.getBasePrice().add(color.getPrice()).
44              add(engine.getPrice()).add(tire.getPrice());
45
46          return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price);
47      }
48
49      /**
50       *
51       * @param orderRequest
52       * @return
53       */
54
55      private List<Option> getOption(OrderRequest orderRequest) {
56          return jeepOrderDao.fetchOptions(orderRequest.getOptions());
57      }
58
```

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();
```

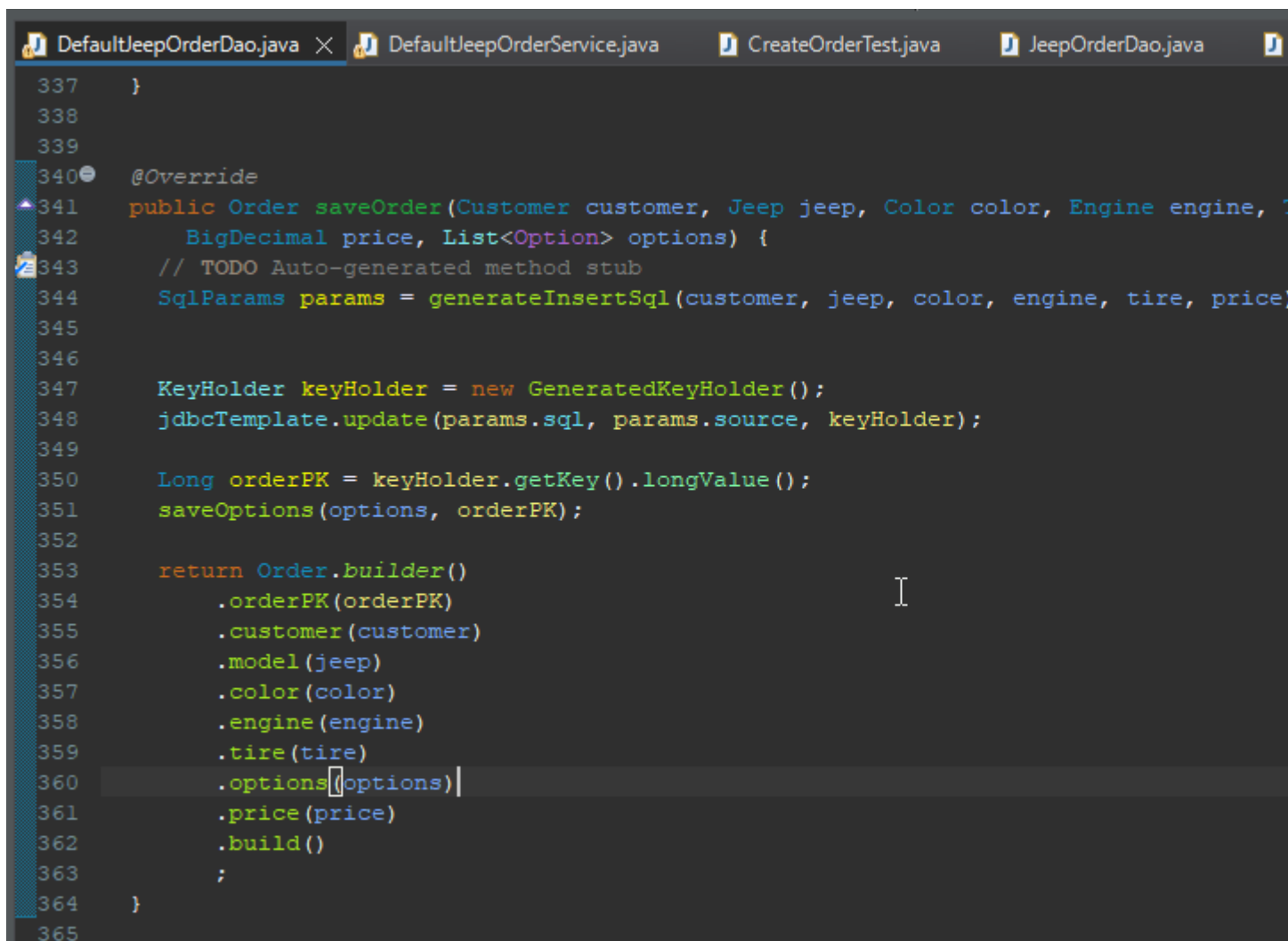
Be sure to extract the order primary key from the KeyHolder object into a variable of type Long named orderPK.

- 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.

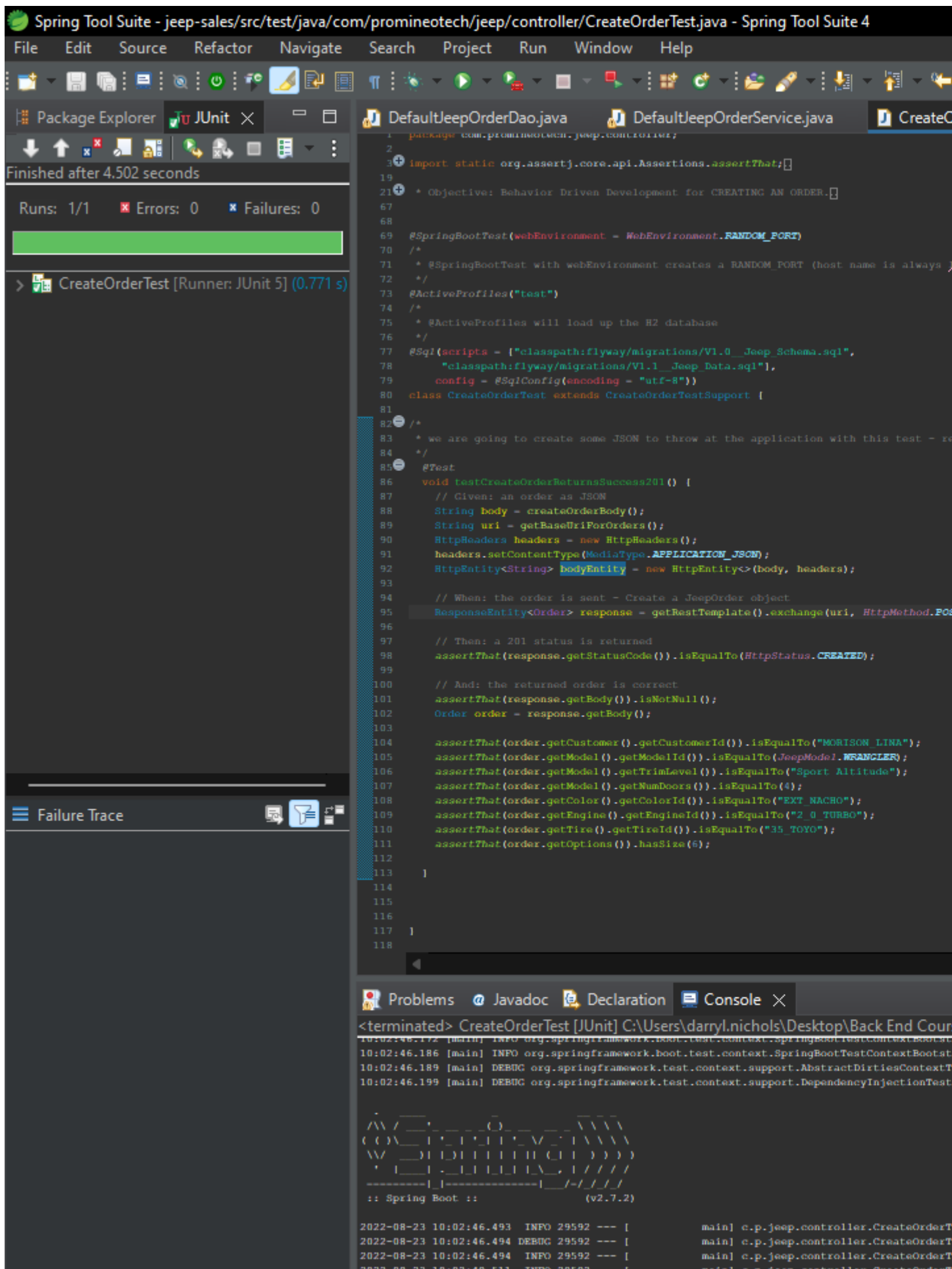
- iv) In the `saveOrder` method in the DAO implementation, return an `Order` object using the `Order.builder`. The `Order` should include `orderPK`, customer, jeep (model), color, engine, tire, options and price.
- v) Produce a screenshot of the `saveOrder` method. 



```
DefaultJeepOrderDao.java x DefaultJeepOrderService.java CreateOrderTest.java JeepOrderDao.java
337     }
338
339
340     @Override
341     public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine,
342         BigDecimal price, List<Option> options) {
343         // TODO Auto-generated method stub
344         SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
345
346
347         KeyHolder keyHolder = new GeneratedKeyHolder();
348         jdbcTemplate.update(params.sql, params.source, keyHolder);
349
350         Long orderPK = keyHolder.getKey().longValue();
351         saveOptions(options, orderPK);
352
353         return Order.builder()
354             .orderPK(orderPK)
355             .customer(customer)
356             .model(jeep)
357             .color(color)
358             .engine(engine)
359             .tire(tire)
360             .options(options)
361             .price(price)
362             .build()
363         ;
364     }
365
```

- 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. 







**Screenshots of Code:**

Found where  is notated

**Screenshots of Running Application:**

Found where  is notated

**URL to GitHub Repository:**

<https://github.com/dnich02f/SpringBoot-Week16-CodingAssignment>