

Web API Design with Spring Boot Week 16 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/AvannR/SpringBoot-Jeep-Sales-Week-13-to-16->


URL to Public Link of your Video: <https://youtu.be/18L78CbrCbA>

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.

- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Use your existing repo or create a new repository on GitHub for this week's assignment and push your completed code to the repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the screenshots into this Assignment Document indicated by: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.


2. In addition, please include the following in your Coding Assignment Document:

- The requested screenshots, indicated by: 
- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-

Web API Design with Spring Boot Week 16 Coding Assignment

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.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 1b. For example:

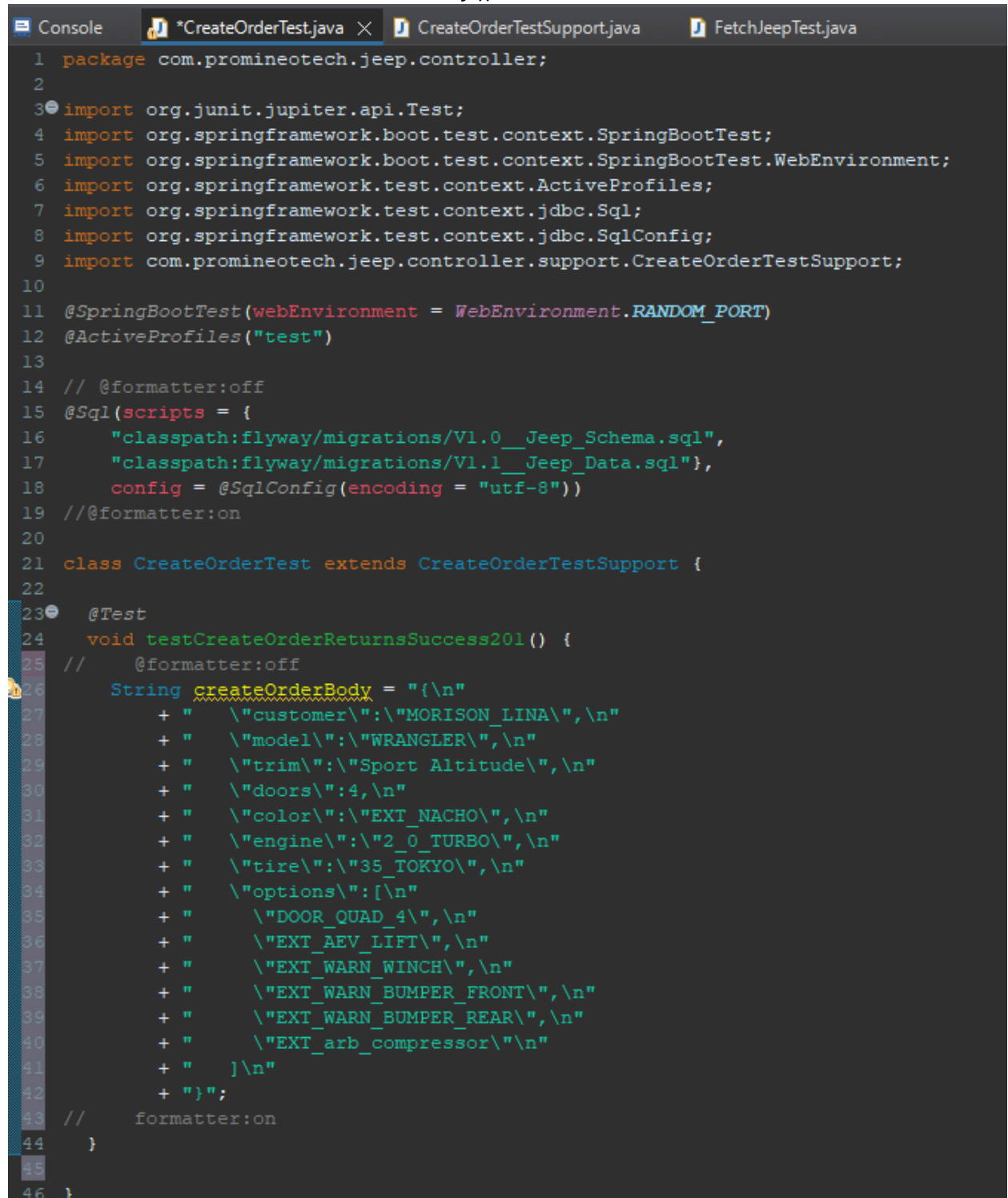
Web API Design with Spring Boot Week 16 Coding Assignment

```
{  
  "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/>.

Web API Design with Spring Boot Week 16 Coding Assignment

Produce a screenshot of the createOrderBody() method. 



```
1 package com.promineotech.jeep.controller;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.boot.test.context.SpringBootTest;
5 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
6 import org.springframework.test.context.ActiveProfiles;
7 import org.springframework.test.context.jdbc.Sql;
8 import org.springframework.test.context.jdbc.SqlConfig;
9 import com.promineotech.jeep.controller.support.CreateOrderTestSupport;
10
11 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
12 @ActiveProfiles("test")
13
14 // @formatter:off
15 @Sql(scripts = {
16     "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
17     "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
18     config = @SqlConfig(encoding = "utf-8"))
19 // @formatter:on
20
21 class CreateOrderTest extends CreateOrderTestSupport {
22
23     @Test
24     void testCreateOrderReturnsSuccess201() {
25         // @formatter:off
26         String createOrderBody = "{\n"
27             + "  \"customer\": \"MORISON_LINA\",\n"
28             + "  \"model\": \"WRANGLER\",\n"
29             + "  \"trim\": \"Sport Altitude\",\n"
30             + "  \"doors\": 4,\n"
31             + "  \"color\": \"EXT_NACHO\",\n"
32             + "  \"engine\": \"2_0_TURBO\",\n"
33             + "  \"tire\": \"35_TOKYO\",\n"
34             + "  \"options\": [\n"
35             + "    \"DOOR_QUAD_4\",\n"
36             + "    \"EXT_AEV_LIFT\",\n"
37             + "    \"EXT_WARN_WINCH\",\n"
38             + "    \"EXT_WARN BUMPER_FRONT\",\n"
39             + "    \"EXT_WARN BUMPER_REAR\",\n"
40             + "    \"EXT_arb_compressor\"\n"
41             + "  ]\n"
42             + "}";
43         // @formatter:on
44     }
45 }
46 }
```

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

Web API Design with Spring Boot Week 16 Coding Assignment

- 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.jeepp.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);
```

Web API Design with Spring Boot Week 16 Coding Assignment

k) Produce a screenshot of the test method. 

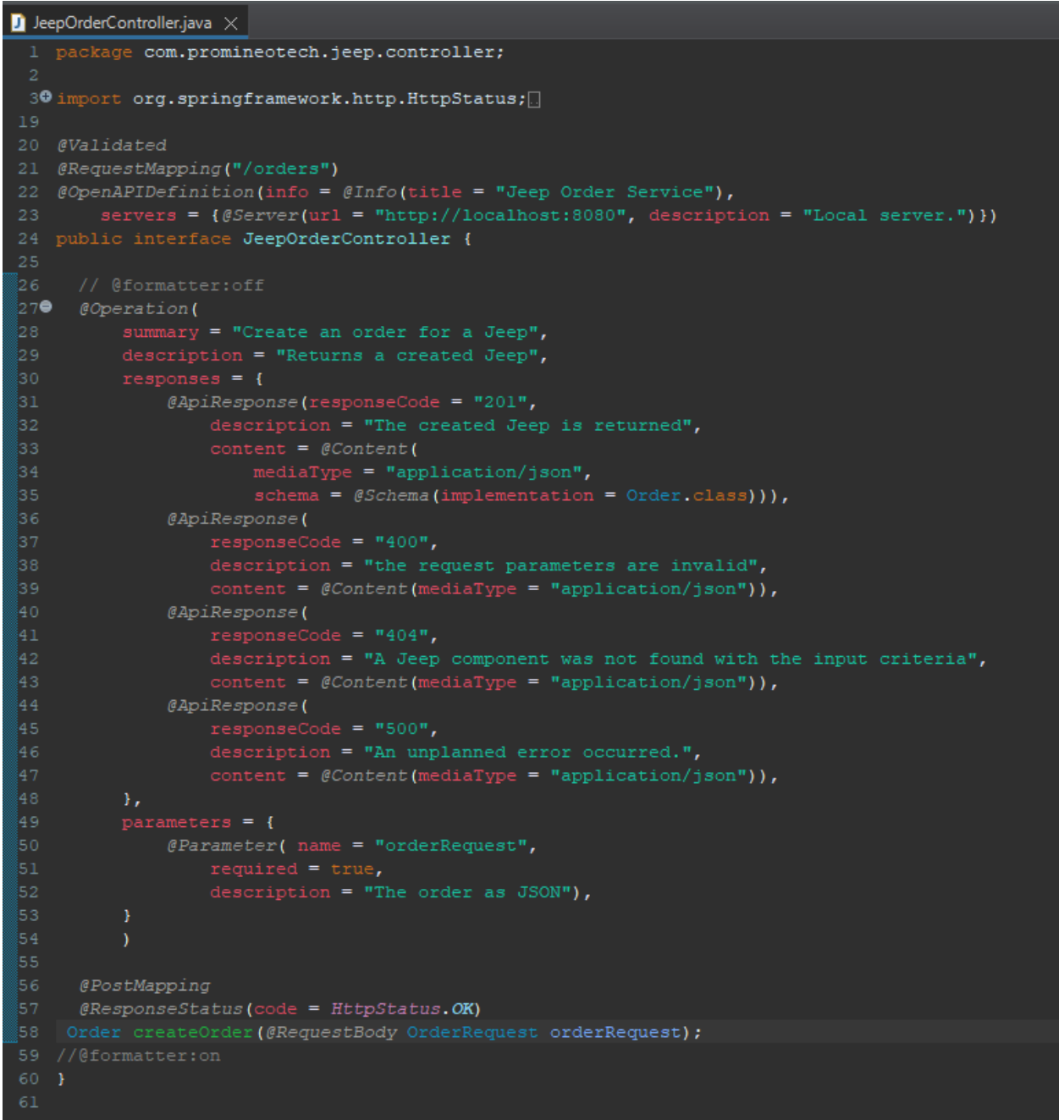
Web API Design with Spring Boot Week 16 Coding Assignment

```
Console CreateOrderTest.java X
35 @Test
36 void testCreateOrderReturnsSuccess201() {
37     // Given: an order as JSON
38     String body = createOrderBody();
39     String uri = String.format("http://localhost:%d/orders", serverPort);
40     HttpHeaders headers = new HttpHeaders();
41     headers.setContentType(MediaType.APPLICATION_JSON);
42     HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
43
44     //When: the order is sent
45     ResponseEntity<Order> response = restTemplate.exchange(uri,
46         HttpMethod.POST, bodyEntity, Order.class);
47
48     //Then: a 201 status is returned
49     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
50
51     //And: the returned order is correct);
52     assertThat(response.getBody()).isNotNull();
53     Order order = response.getBody();
54     assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
55     assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
56     assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
57     assertThat(order.getModel().getNumDoors()).isEqualTo(4);
58     assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");
59     assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
60     assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
61     assertThat(order.getOptions()).hasSize(6);
62
63 }
64
65 protected String createOrderBody() {
66     // @formatter:off
67     return "{\n"
68         + "  \"customer\": \"MORISON_LINA\", \n"
69         + "  \"model\": \"WRANGLER\", \n"
70         + "  \"trim\": \"Sport Altitude\", \n"
71         + "  \"doors\": 4, \n"
72         + "  \"color\": \"EXT_NACHO\", \n"
73         + "  \"engine\": \"2_0_TURBO\", \n"
74         + "  \"tire\": \"35_TOYO\", \n"
75         + "  \"options\": [\n"
76         + "    \"DOOR_QUAD_4\", \n"
77         + "    \"EXT_AEV_LIFT\", \n"
78         + "    \"EXT_WARN_WINCH\", \n"
79         + "    \"EXT_WARN BUMPER_FRONT\", \n"
80         + "    \"EXT_WARN BUMPER_REAR\", \n"
81         + "    \"EXT_arb_compressor\" \n"
82         + "  ] \n"
83         + "}";
84     // @formatter:on
85 }
86
87 }
```

Web API Design with Spring Boot Week 16 Coding Assignment

- 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.

Web API Design with Spring Boot Week 16 Coding Assignment

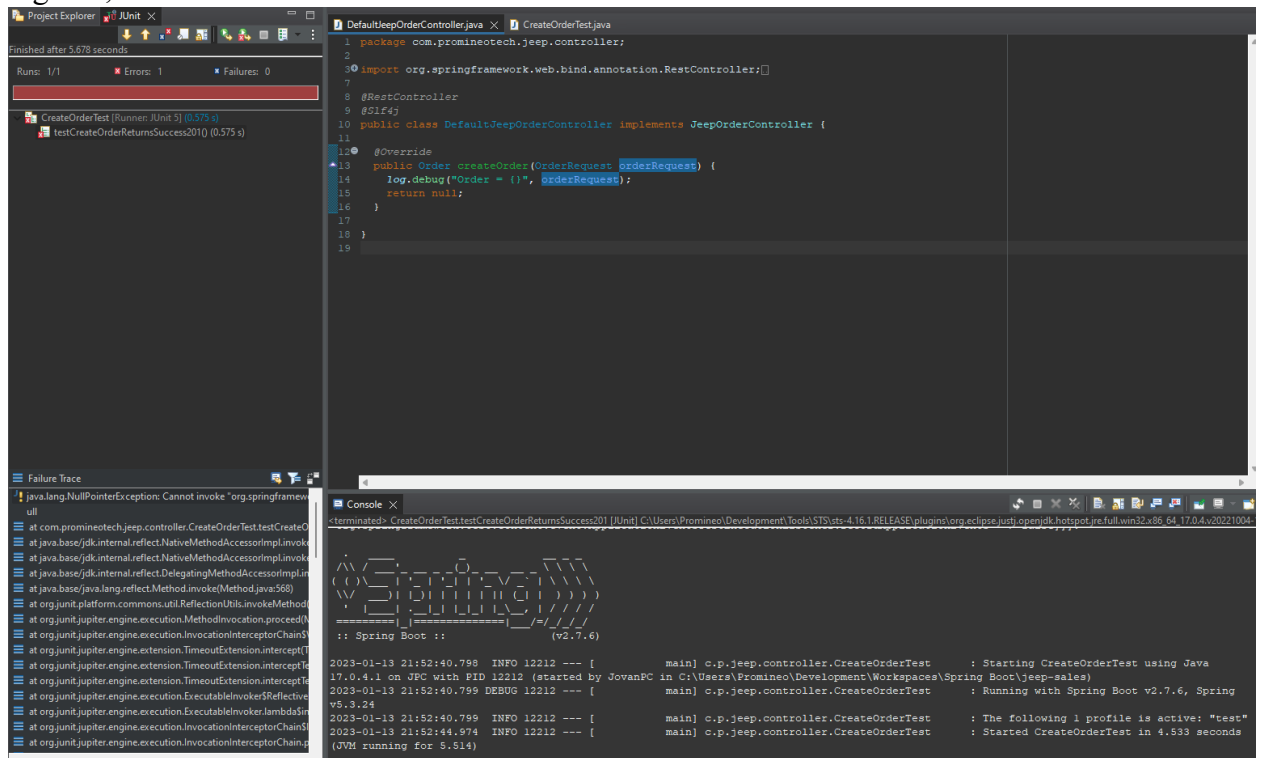
- c) Produce a screenshot of the finished JeepOrderController interface showing no compile errors. 

```
1 package com.promineotech.jeep.controller;
2
3 import org.springframework.http.HttpStatus;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 @Validated
21 @RequestMapping("/orders")
22 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"),
23     servers = {@Server(url = "http://localhost:8080", description = "Local server.")})
24 public interface JeepOrderController {
25
26     // @formatter:off
27     @Operation(
28         summary = "Create an order for a Jeep",
29         description = "Returns a created Jeep",
30         responses = {
31             @ApiResponse(responseCode = "201",
32                 description = "The created Jeep is returned",
33                 content = @Content(
34                     mediaType = "application/json",
35                     schema = @Schema(implementation = Order.class))),
36             @ApiResponse(
37                 responseCode = "400",
38                 description = "the request parameters are invalid",
39                 content = @Content(mediaType = "application/json")),
40             @ApiResponse(
41                 responseCode = "404",
42                 description = "A Jeep component was not found with the input criteria",
43                 content = @Content(mediaType = "application/json")),
44             @ApiResponse(
45                 responseCode = "500",
46                 description = "An unplanned error occurred.",
47                 content = @Content(mediaType = "application/json")),
48         },
49         parameters = {
50             @Parameter(name = "orderRequest",
51                 required = true,
52                 description = "The order as JSON"),
53         }
54     )
55
56     @PostMapping
57     @ResponseStatus(code = HttpStatus.OK)
58     Order createOrder(@RequestBody OrderRequest orderRequest);
59 // @formatter:on
60 }
61
```

- 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)

Web API Design with Spring Boot Week 16 Coding Assignment

- 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.



- 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)`

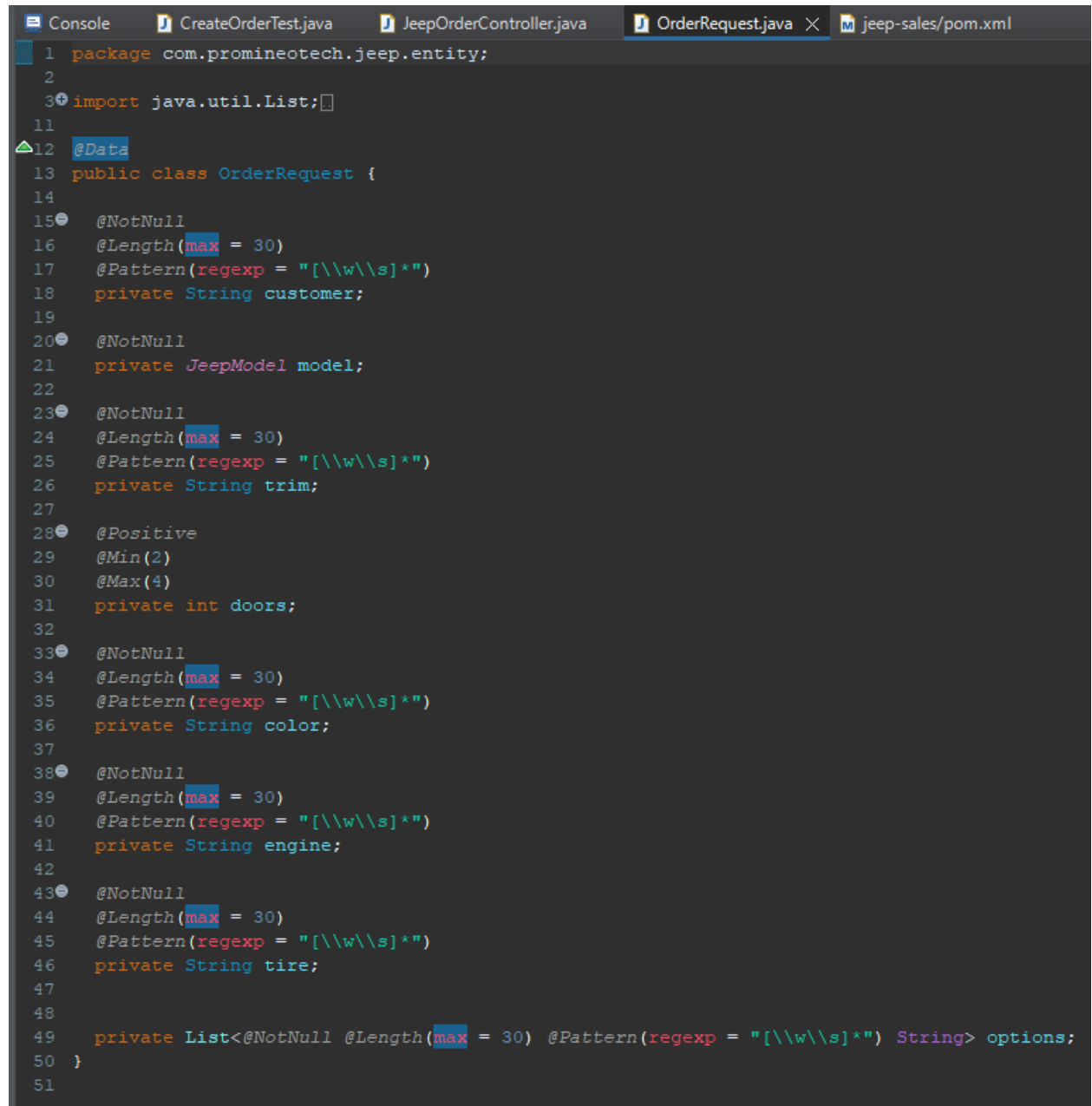
Web API Design with Spring Boot Week 16 Coding Assignment

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

Web API Design with Spring Boot Week 16 Coding Assignment

- 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).

The screenshot displays an IDE with several components:

- Project Explorer:** Shows the project structure with a package `jeep.service` containing `JeepOrderService.java` and `DefaultJeepOrderService.java`.
- Code Editor:** Shows the implementation of `DefaultJeepOrderService.java`. It includes a `createOrder` method that logs the `orderRequest` parameter using `logger.info("createOrder: " + orderRequest);` before returning the result.
- Console:** Shows the output of the test `CreateOrderTest`. It includes the following log lines:

```
2023-01-13 22:38:44.419 INFO 18676 --- [main] o.p.jee.controller.CreateOrderTest : Starting CreateOrderTest using Java 17.0.4.1 on JFC with PID 18676 (started by JovanPC in C:\Users\Promineo\Development\workspace\Spring Boot\jeep-sales)
2023-01-13 22:38:44.419 DEBUG 18676 --- [main] o.p.jee.controller.CreateOrderTest : Running with Spring Boot v2.7.6, Spring v5.3.24
2023-01-13 22:38:44.420 INFO 18676 --- [main] o.p.jee.controller.CreateOrderTest : The following 1 profile is active: "test"
2023-01-13 22:38:48.785 INFO 18676 --- [main] o.p.jee.controller.CreateOrderTest : Started CreateOrderTest in 4.749 seconds (JVM running for 5.678)
```

- 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.

Web API Design with Spring Boot Week 16 Coding Assignment

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.

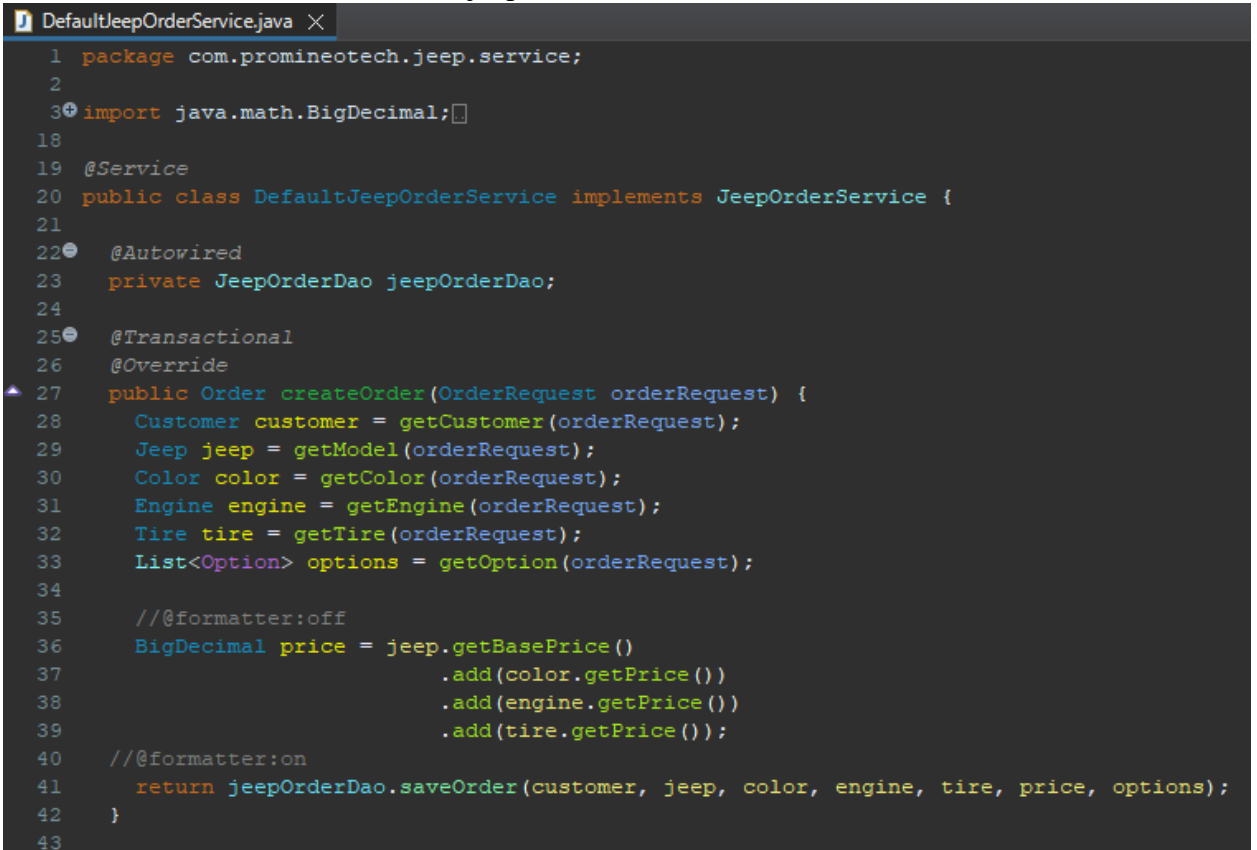
14) In `DefaultJeepOrderService.java`, work with the method `createOrder`.

- 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);
```

Web API Design with Spring Boot Week 16 Coding Assignment

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

```
DefaultJeepOrderService.java X
1 package com.promineotech.jeep.service;
2
3 import java.math.BigDecimal;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 @Service
20 public class DefaultJeepOrderService implements JeepOrderService {
21
22     @Autowired
23     private JeepOrderDao jeepOrderDao;
24
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         //@formatter:off
36         BigDecimal price = jeep.getBasePrice()
37             .add(color.getPrice())
38             .add(engine.getPrice())
39             .add(tire.getPrice());
40         //@formatter:on
41         return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
42     }
43 }
```

- 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:


Web API Design with Spring Boot Week 16 Coding Assignment

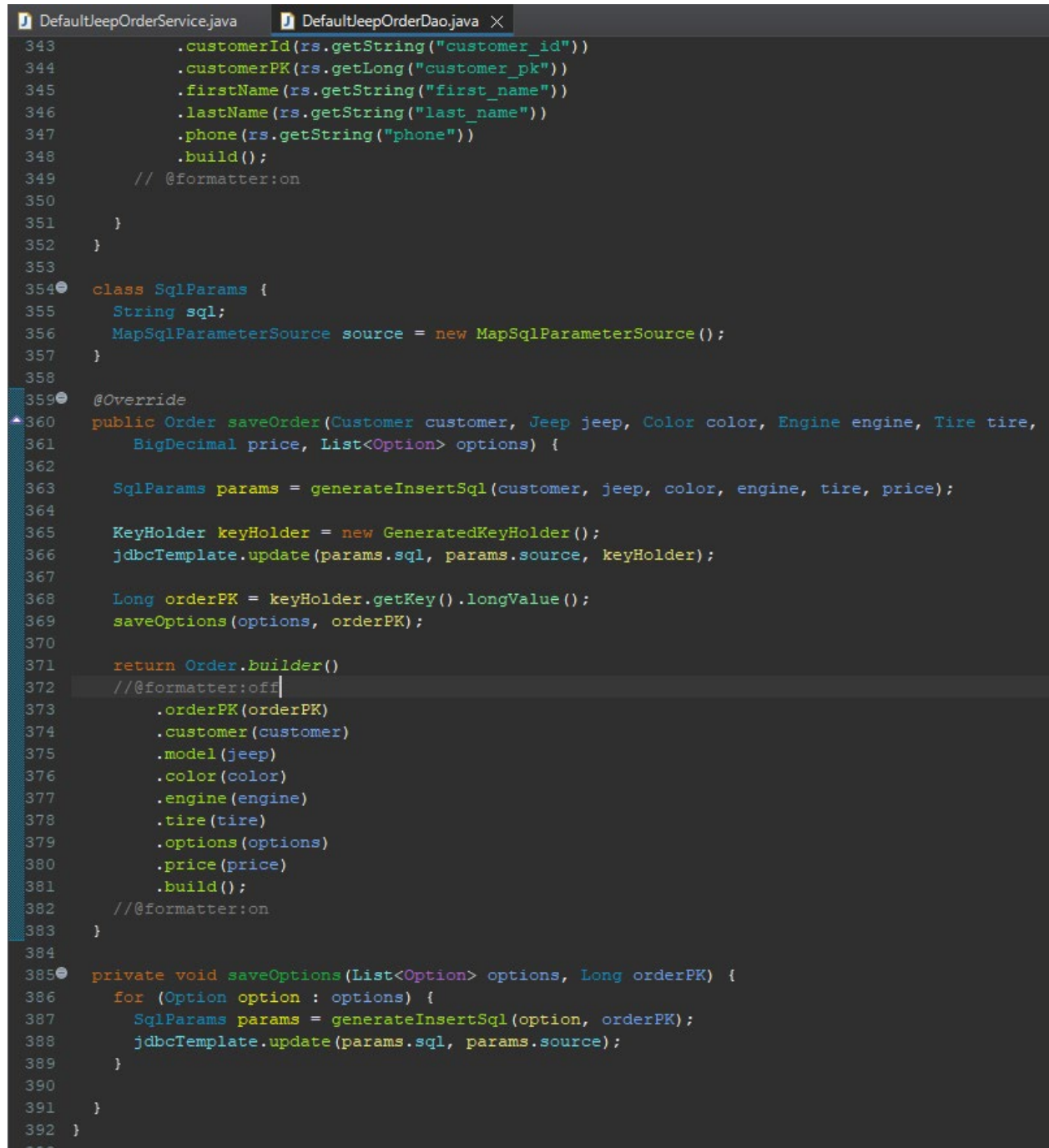
```
private void saveOptions(List<Option> options, Long orderPK)
```

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`.


Web API Design with Spring Boot Week 16 Coding Assignment

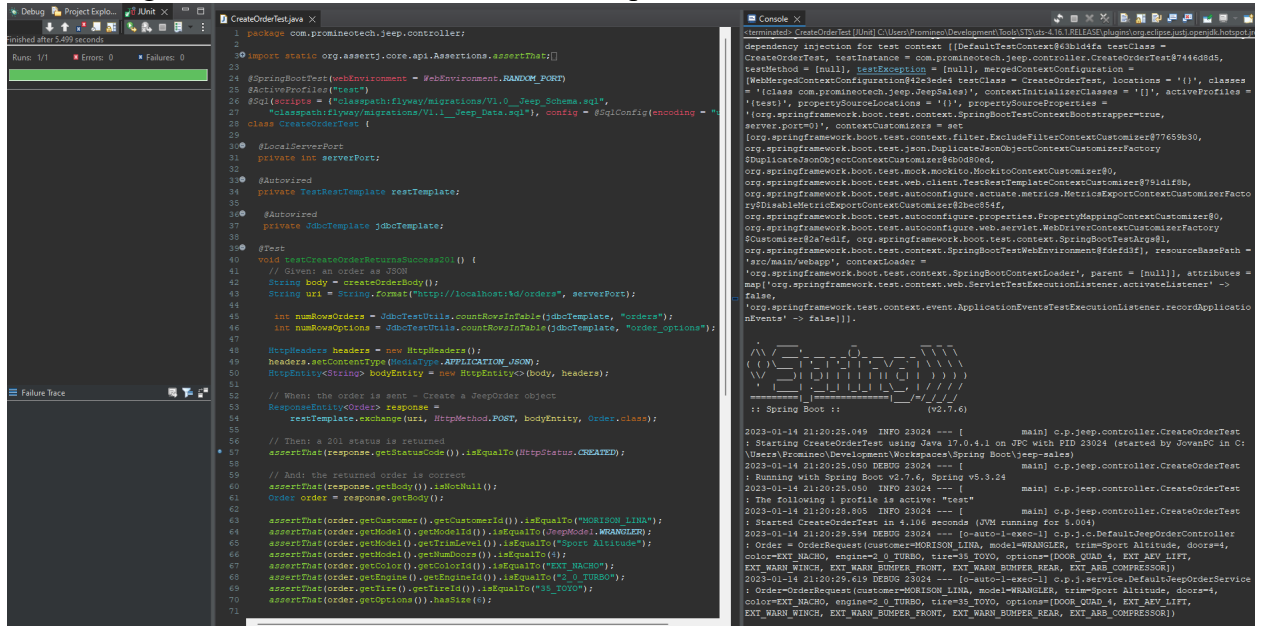
v) Produce a screenshot of the saveOrder method. 



```
DefaultJeepOrderService.java  DefaultJeepOrderDao.java X
343         .customerId(rs.getString("customer_id"))
344         .customerPK(rs.getLong("customer_pk"))
345         .firstName(rs.getString("first_name"))
346         .lastName(rs.getString("last_name"))
347         .phone(rs.getString("phone"))
348         .build();
349         // @formatter:on
350
351     }
352 }
353
354 class SqlParams {
355     String sql;
356     MapSqlParameterSource source = new MapSqlParameterSource();
357 }
358
359 @Override
360 public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
361     BigDecimal price, List<Option> options) {
362
363     SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
364
365     KeyHolder keyHolder = new GeneratedKeyHolder();
366     jdbcTemplate.update(params.sql, params.source, keyHolder);
367
368     Long orderPK = keyHolder.getKey().longValue();
369     saveOptions(options, orderPK);
370
371     return Order.builder()
372         // @formatter:off
373         .orderPK(orderPK)
374         .customer(customer)
375         .model(jeep)
376         .color(color)
377         .engine(engine)
378         .tire(tire)
379         .options(options)
380         .price(price)
381         .build();
382         // @formatter:on
383     }
384
385 private void saveOptions(List<Option> options, Long orderPK) {
386     for (Option option : options) {
387         SqlParams params = generateInsertSql(option, orderPK);
388         jdbcTemplate.update(params.sql, params.source);
389     }
390 }
391
392 }
```


Web API Design with Spring Boot Week 16 Coding Assignment

- 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. 



The screenshot displays an IDE with two main panels. The left panel shows the `CreateOrderTest.java` file, which is a JUnit 5 integration test for the `CreateOrder` endpoint. The test class is annotated with `@SpringBootTest` and `@SqlConfig`. It includes a `testCreateOrderReturnsSuccess201()` method that sends a POST request to `/orders` with a JSON body representing a car order. The test uses `assertThat` to verify the response status is 201, the response body is not null, and the order details (customer, model, trim, color, engine, and time) are correct. The right panel shows the console output, which includes the JUnit 5 test runner's progress bar and the test results. The test `testCreateOrderReturnsSuccess201()` passed successfully, as indicated by the green status bar and the 'PASS' result in the console.

```
1 package com.promineotech.jeeb.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
6 @ActiveProfiles("test")
7 @Sql(scripts = {"classpath:flyway/migrations/V1.0_jeep_schema.sql",
8               "classpath:flyway/migrations/V1.1_jeep_data.sql"}, config = @SqlConfig(encoding = "UTF-8"))
9 class CreateOrderTest {
10
11     @LocalServerPort
12     private int serverPort;
13
14     @Autowired
15     private TestRestTemplate restTemplate;
16
17     @Autowired
18     private JdbcTemplate jdbcTemplate;
19
20     @Test
21     void testCreateOrderReturnsSuccess201() {
22         // Given an order as JSON
23         String body = createOrderBody();
24         String url = String.format("http://localhost:%d/orders", serverPort);
25
26         int numberOfOrders = jdbcTemplate.countRowsInTable(jdbcTemplate, "orders");
27         int numberOfOptions = jdbcTemplate.countRowsInTable(jdbcTemplate, "order_options");
28
29         HttpHeaders headers = new HttpHeaders();
30         headers.setContentType(MediaType.APPLICATION_JSON);
31         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
32
33         // When the order is sent - Create a JeepOrder object
34         ResponseEntity<Order> response =
35             restTemplate.exchange(url, HttpMethod.POST, bodyEntity, Order.class);
36
37         // Then a 201 status is returned
38         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
39
40         // And the returned order is correct
41         assertThat(response.getBody()).isNotNull();
42         Order order = response.getBody();
43
44         assertThat(order.getCustomerId()).isEqualTo("MORISON_LINA");
45         assertThat(order.getModelId()).isEqualTo(JeepModel.WRANGLER);
46         assertThat(order.getTrimLevel()).isEqualTo("Sport Altitude");
47         assertThat(order.getModel().getNumberOfDoors()).isEqualTo(4);
48         assertThat(order.getColorId()).isEqualTo("EXT_BLACK");
49         assertThat(order.getEngineId()).isEqualTo("2.0_TURBO");
50         assertThat(order.getTimeId()).isEqualTo("35_TOTO");
51         assertThat(order.getOptions()).hasSize(3);
52     }
53 }
```

```
2023-01-14 21:20:25.049 INFO 23024 --- [main] c.p.jeeb.controller.CreateOrderTest : Starting CreateOrderTest using Java 17.0.4.1 on SPC with PID 23024 (started by JovanPC in C:\Users\JovanPC\AppData\Local\Temp\Spring Boot\jeep-sales)
2023-01-14 21:20:25.050 DEBUG 23024 --- [main] c.p.jeeb.controller.CreateOrderTest : Running with Spring Boot v2.7.6, Spring v5.3.24
2023-01-14 21:20:25.050 INFO 23024 --- [main] c.p.jeeb.controller.CreateOrderTest : The following 1 profile is active: "test"
2023-01-14 21:20:28.805 INFO 23024 --- [main] c.p.jeeb.controller.CreateOrderTest : Started CreateOrderTest in 4.106 seconds (JVM running for 3.804)
2023-01-14 21:20:29.394 DEBUG 23024 --- [auto-1-exec-1] c.p.c.DefaultJeepOrderController : Order = OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport Altitude, doors=4, color=EXT_BLACK, engine=2.0 TURBO, time=35_TOTO, options=[DOOR_QUAD_4, EXT_ABY_LIFT, EXT_WARN_WINCH, EXT_WARN_BUMPER_FRONT, EXT_WARN_BUMPER_REAR, EXT_ABY_COMPRESSOR])
2023-01-14 21:20:29.619 DEBUG 23024 --- [auto-1-exec-1] c.p.jeeb.service.DefaultJeepOrderService : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport Altitude, doors=4, color=EXT_BLACK, engine=2.0 TURBO, time=35_TOTO, options=[DOOR_QUAD_4, EXT_ABY_LIFT, EXT_WARN_WINCH, EXT_WARN_BUMPER_FRONT, EXT_WARN_BUMPER_REAR, EXT_ABY_COMPRESSOR])
```