

Web API Design with Spring Boot Week 16 Coding Assignment


Points possible: 75

URL to GitHub Repository: [kmaradiaga18/Week13IntroSpringBoot](https://github.com/kmaradiaga18/Week13IntroSpringBoot): 1st week of SpringBoot (github.com)


URL to Public Link of your Video:

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.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 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/>.

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

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

```
void testCreateOrderReturnSuccess201() {
    String body = createOrderBody();

}

/**
 *
 */
protected String createOrderBody() {
    //@formatter:off
    return "{\r\n"
        + "  \"customer\": \"MORISON_LINA\", \r\n"
        + "  \"model\": \"WRANGLER\", \r\n"
        + "  \"trim\": \"Sport Altitude\", \r\n"
        + "  \"doors\": 4, \r\n"
        + "  \"color\": \"EXT_NACHO\", \r\n"
        + "  \"engine\": \"2_0_TURBO\", \r\n"
        + "  \"tire\": \"35_TOYO\", \r\n"
        + "  \"options\": [\r\n"
        + "    \"DOOR_QUAD_4\", \r\n"
        + "    \"EXT_AEV_LIFT\", \r\n"
        + "    \"EXT_WARN_WINCH\", \r\n"
        + "    \"EXT_WARN BUMPER_FRONT\", \r\n"
        + "    \"EXT_WARN BUMPER_REAR\", \r\n"
        + "    \"EXT_ARB_COMPRESSOR\" \r\n"
        + "  ] \r\n"
        + "}";
    //@formatter:off
}
```

- 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);
```

Web API Design with Spring Boot Week 16 Coding Assignment

- 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. 


Web API Design with Spring Boot Week 16 Coding Assignment

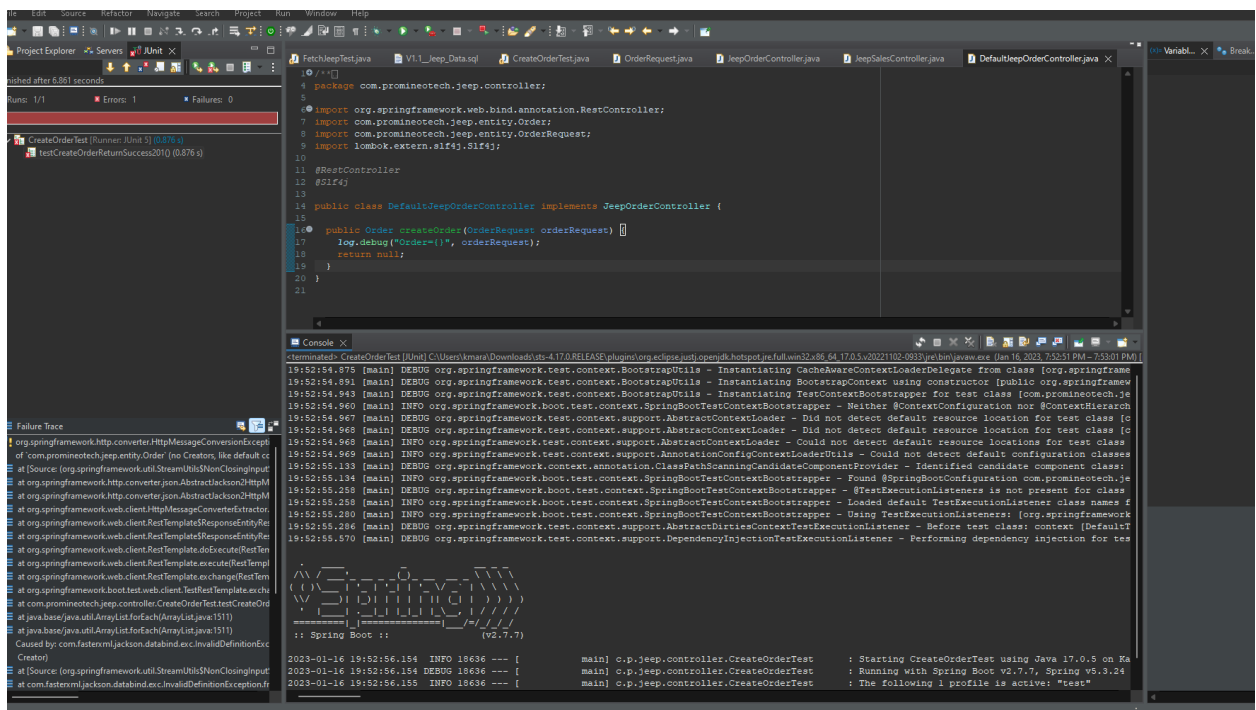
```
36
37 TestRestTemplate restTemplate = new TestRestTemplate();
38
39 @Test
40 void testCreateOrderReturnSuccess201() {
41     //Given: an order as JSON
42     String body = createOrderBody();
43     String uri = String.format("http://localhost:%d/orders", serverPort);
44
45     HttpHeaders headers = new HttpHeaders();
46     headers.setContentType(MediaType.APPLICATION_JSON);
47
48     HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
49
50     //When: the order is sent
51     ResponseEntity<Order> response = restTemplate.exchange(uri,
52         HttpMethod.POST, bodyEntity, Order.class);
53
54     //Then: a 201 status is returned
55     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
56     assertThat(response.getBody()).isNotNull();
57
58     Order order = response.getBody();
59     assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
60     assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
61     assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
62     assertThat(order.getModel().getNumDoors()).isEqualTo(4);
63     assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");
64     assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
65     assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
66     assertThat(order.getOptions()).hasSize(6);
67
68
69     //And: the returned order is correct
```

- 3) In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add @RequestMapping("/orders") as a class-level annotation.
- 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).
 - Add the @RequestBody annotation to the orderRequest parameter. Make sure to add the RequestBody annotation from the org.springframework.web.bind.annotation package.
 - Produce a screenshot of the finished JeepOrderController interface showing no compile errors. 

Web API Design with Spring Boot Week 16 Coding Assignment

```
13 // **
14 * @author kmara
15 *
16 */
17
18 @Validated
19 @RequestMapping("/orders")
20 public interface JeepOrderController {
21     @PostMapping
22     @ResponseStatus(code = HttpStatus.CREATED)
23     Order createOrder(@Valid @RequestBody OrderRequest orderRequest);
24 }
25
```

- 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. 




- 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).

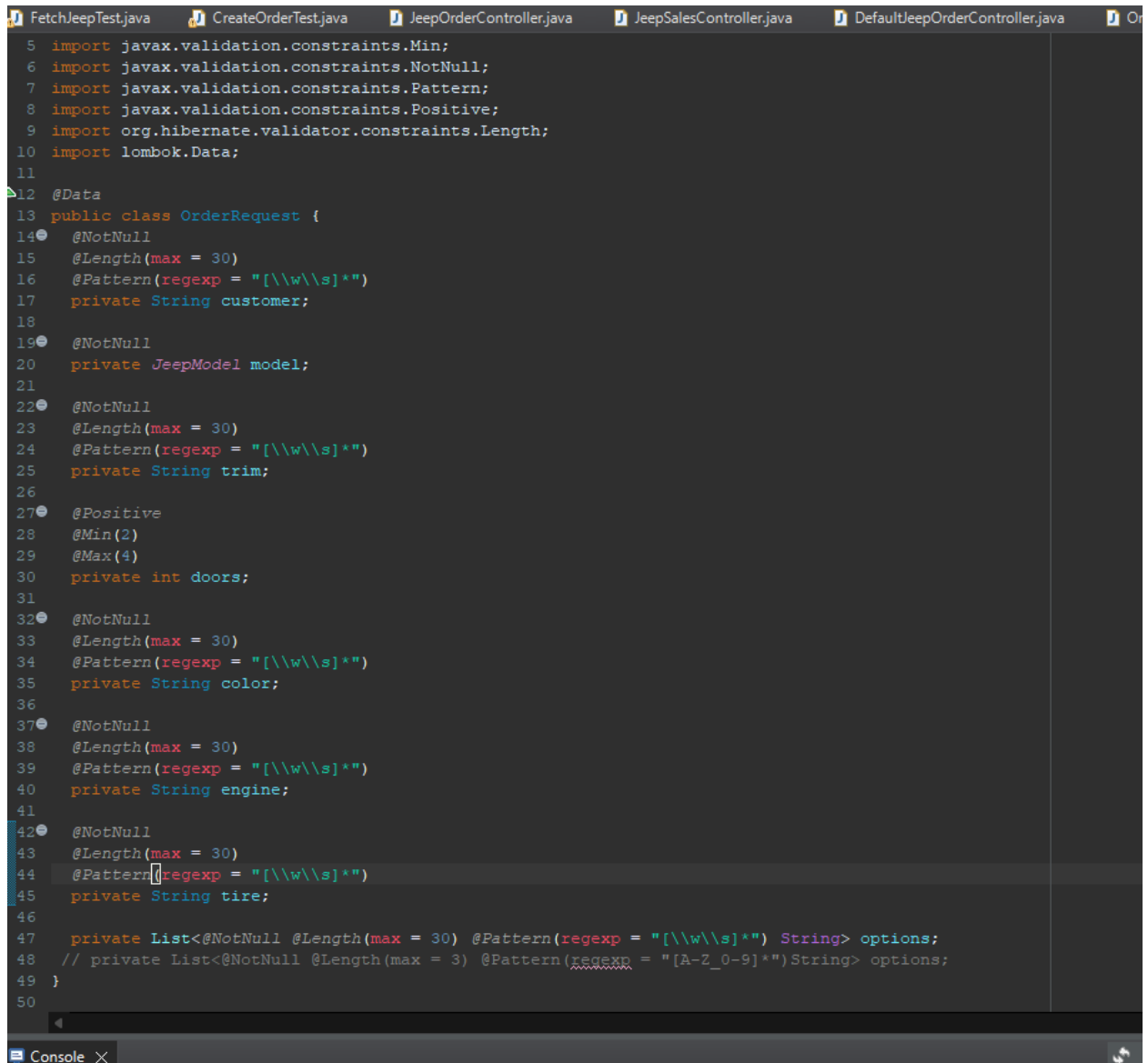
Web API Design with Spring Boot Week 16 Coding Assignment

- 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(regex = "[\\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(regex = "[\\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. 


Web API Design with Spring Boot Week 16 Coding Assignment



```
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     // private List<@NotNull @Length(max = 3) @Pattern(regexp = "[A-Z_0-9]*")String> options;
49 }
50
```

- 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.

Web API Design with Spring Boot Week 16 Coding Assignment

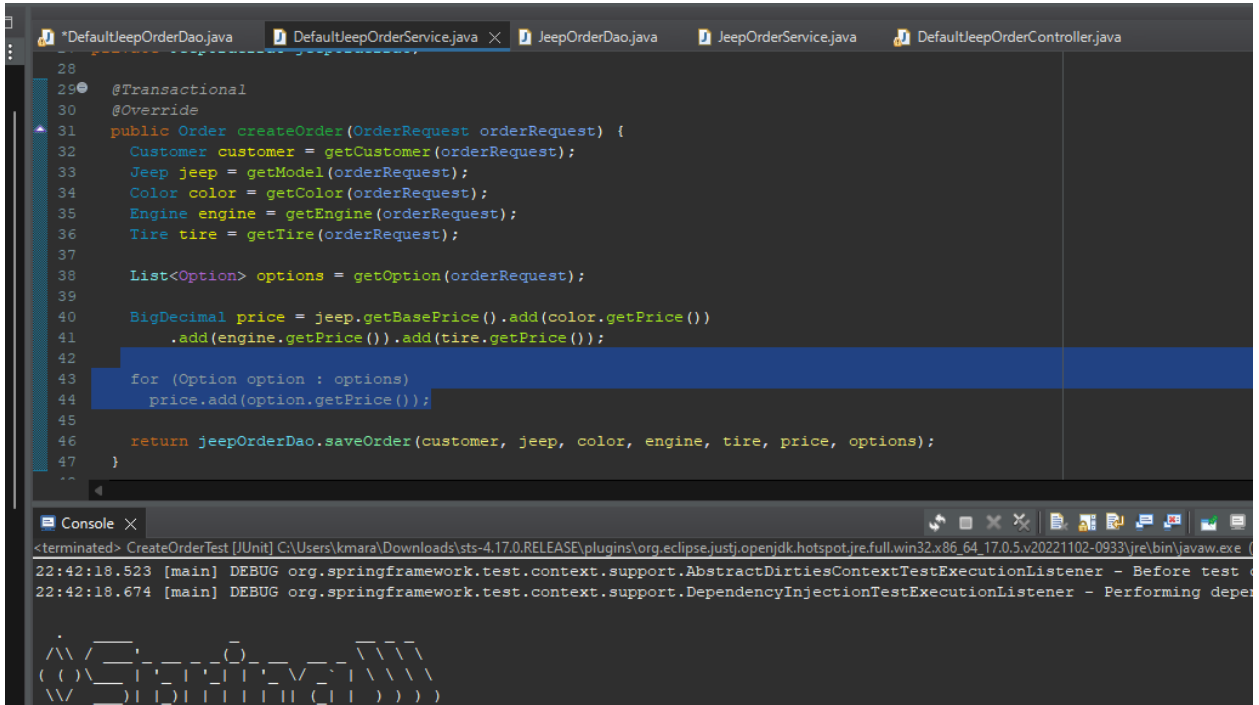
- f) Run the test `CreateOrderTest` again. Produce a screenshot showing that the service layer `createOrder` method correctly prints the log line in the console. (e.g. prints out the `OrderRequest` in the console from within the Service Layer). 
- 9) In the `jeep.dao` sub-package, create the empty (no 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.

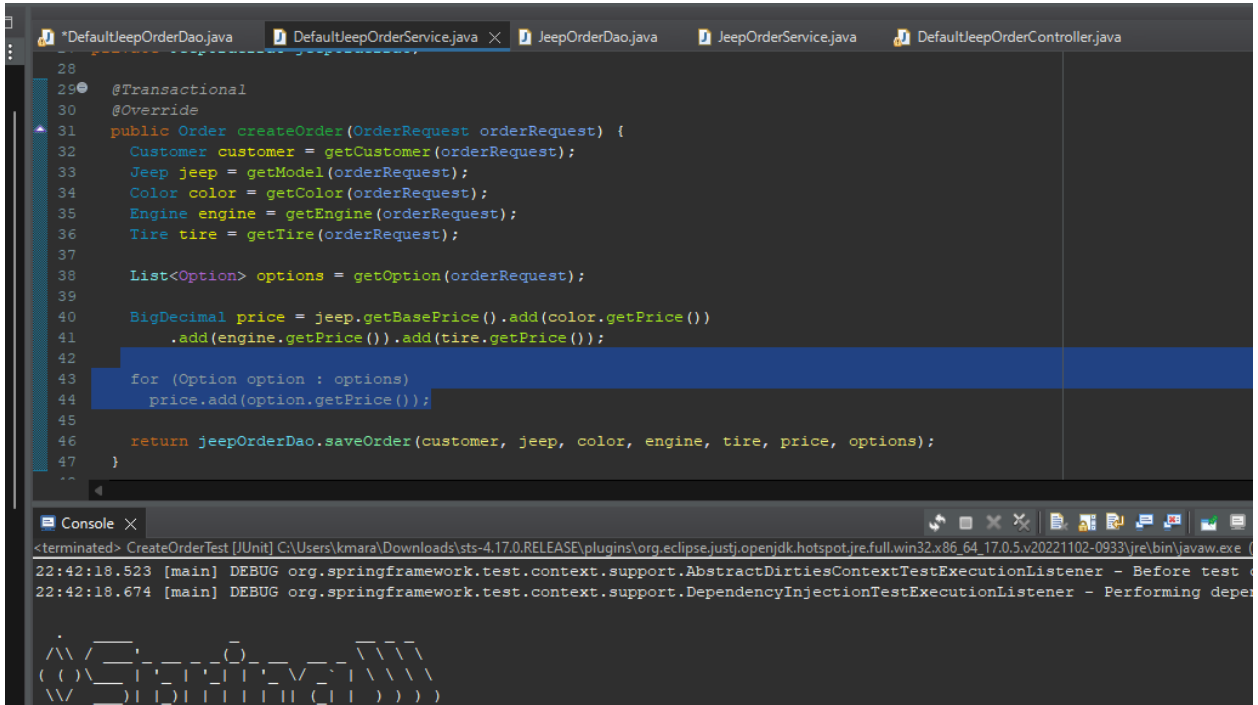
Web API Design with Spring Boot Week 16 Coding Assignment

- 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. 

The screenshot shows an IDE with several Java files open. The active file is DefaultJeepOrderService.java, showing the createOrder method. The method is annotated with @Transactional and @Override. It calls getCustomer, getModel, getColor, getEngine, getTire, and getOption to retrieve data from an OrderRequest. It then calculates the total price by summing the base price of the jeep, the price of the color, engine, and tire, and the prices of all options. Finally, it calls jeepOrderDao.saveOrder with the customer, jeep, color, engine, tire, price, and options. The console output at the bottom shows debug messages from Spring Framework test execution listeners.

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

Web API Design with Spring Boot Week 16 Coding Assignment


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. 

```
47  @Override
48  public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
49      BigDecimal price, List<Option> options) {
50
51      SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
52      // return null;
53
54      KeyHolder keyHolder = new GeneratedKeyHolder();
55      jdbcTemplate.update(params.sql, params.source, keyHolder);
56
57      Long orderPK = keyHolder.getKey().longValue();
58      saveOptions(options, orderPK);
59
60      return Order.builder()
61          .orderPK(orderPK)
62          .customer(customer)
63          .model(jeep)
64          .color(color)
65          .engine(engine)
66          .tire(tire)
67          .options(options)
68          .price(price)
69          .build();
70  }
71
72  /**
73   * @param options
74   * @param orderPK
```

- 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. 