


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.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);
```

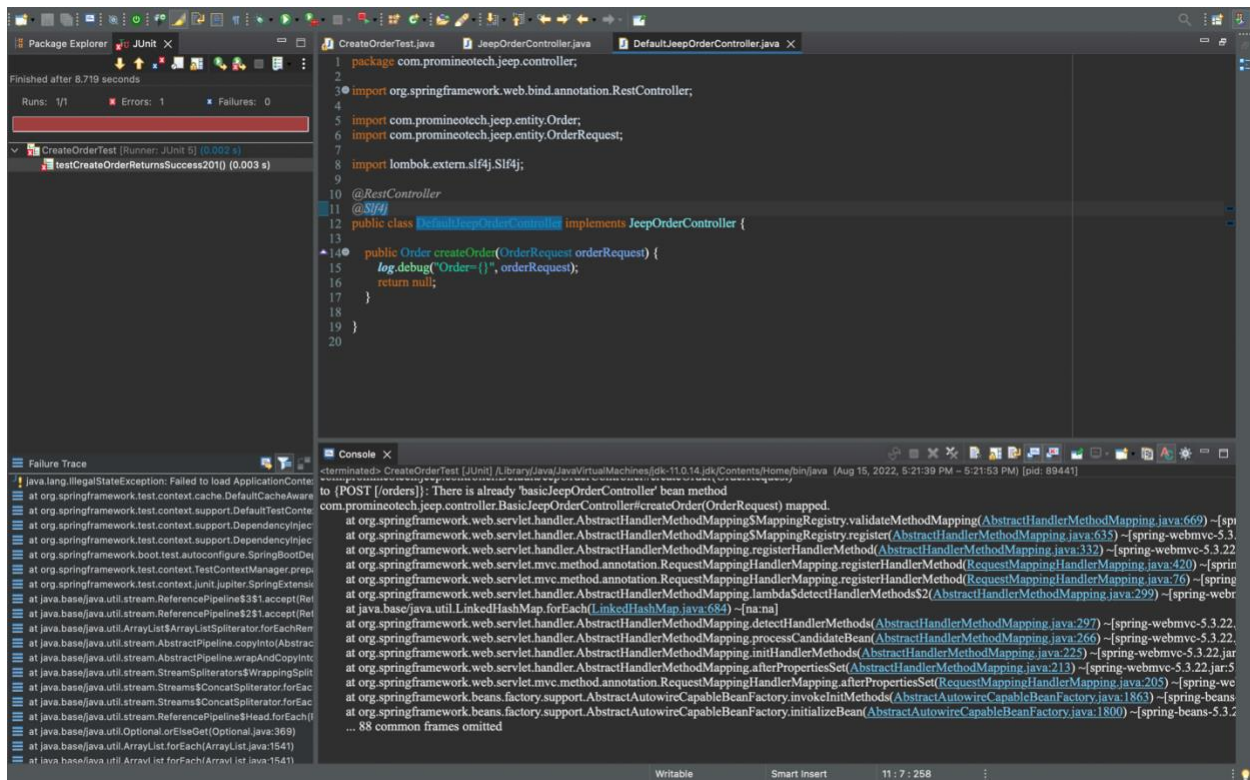
k) Produce a screenshot of the test method. 

```
1 package com.promineotech.jeepp.controller.support;
2
3 public class CreateOrderTestSupport extends BaseTest {
4
5     protected String createOrderBody() {
6
7         return "{\n"
8             + "  \"customer\": \"MORISON_LINA\", \n"
9             + "  \"model\": \"WRANGLER\", \n"
10            + "  \"trim\": \"Sport Altitude\", \n"
11            + "  \"doors\": 4, \n"
12            + "  \"color\": \"EXT_NACHO\", \n"
13            + "  \"engine\": \"2.0 TURBO\", \n"
14            + "  \"tire\": \"35_TOYO\", \n"
15            + "  \"options\": [\n"
16            + "    \"DOOR_QUAD_4\", \n"
17            + "    \"EXT_AEV_LIFT\", \n"
18            + "    \"EXT_WARN_WINCH\", \n"
19            + "    \"EXT_WARN BUMPER_FRONT\", \n"
20            + "    \"EXT_WARN BUMPER_REAR\", \n"
21            + "    \"EXT_ARB_COMPRESSOR\" \n"
22            + "  ] \n"
23            + "}";
24
25     }
26 }
27
```

- 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. 

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


- 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. 

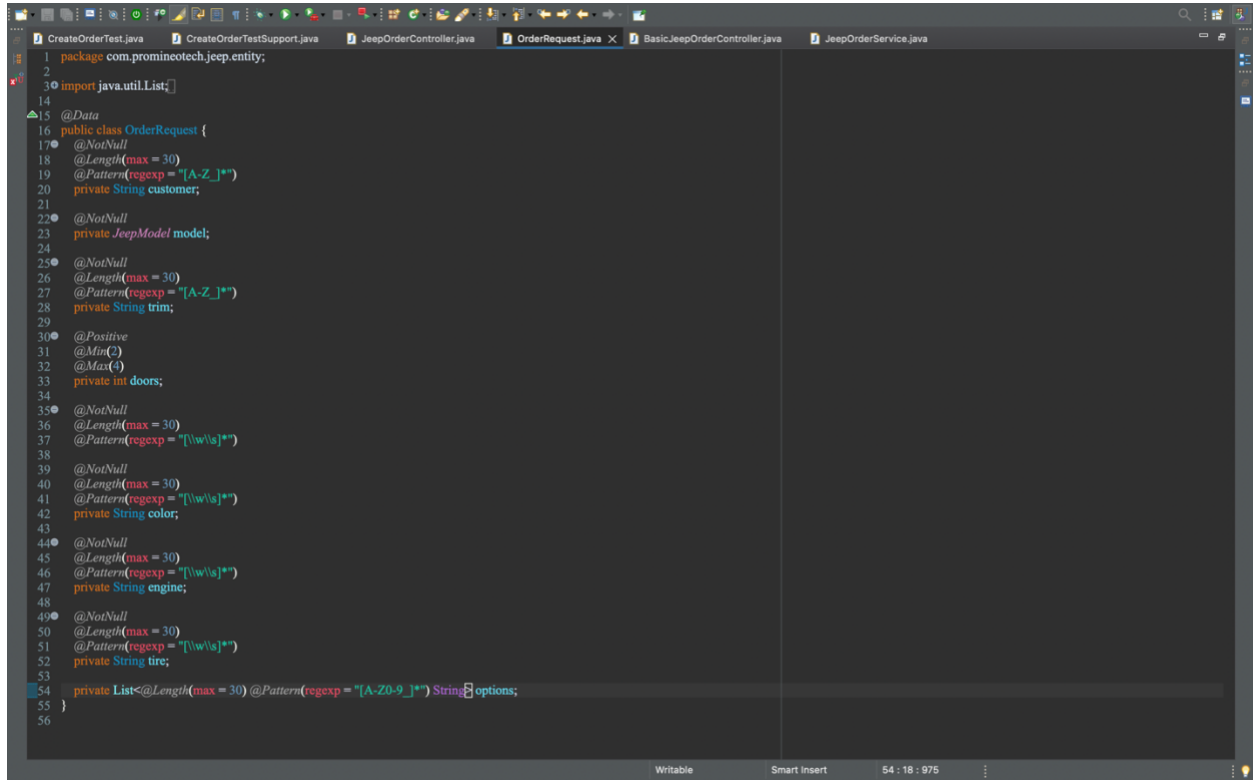


- 5) Find the Maven dependency `spring-boot-starter-validation` by looking it up at <https://mvnrepository.com/>. Add this repository to the project POM file (pom.xml).
- 6) Add the class-level annotation `@Validated` to the `JeepOrderController` interface.
- 7) Add Bean Validation annotations to the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


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


Do not apply a `@NotNull` annotation to the `List` because if you have no options the `List` may be null.

e) Produce a screenshot of this class with the annotations. 




```
1 package com.promineotech.jee.entity;
2
3 import java.util.List;
4
5 @Data
6 public class OrderRequest {
7     @NotNull
8     @Length(max = 30)
9     @Pattern(regexp = "[A-Z_]*")
10    private String customer;
11
12    @NotNull
13    private JeepModel model;
14
15    @NotNull
16    @Length(max = 30)
17    @Pattern(regexp = "[A-Z_]*")
18    private String trim;
19
20    @Positive
21    @Min(2)
22    @Max(4)
23    private int doors;
24
25    @NotNull
26    @Length(max = 30)
27    @Pattern(regexp = "[\\w\\s]*")
28    private String color;
29
30    @NotNull
31    @Length(max = 30)
32    @Pattern(regexp = "[\\w\\s]*")
33    private String engine;
34
35    @NotNull
36    @Length(max = 30)
37    @Pattern(regexp = "[\\w\\s]*")
38    private String tire;
39
40    private List<@Length(max = 30) @Pattern(regexp = "[A-Z0-9_]*") String> options;
41
42 }
```

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). 

- 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.

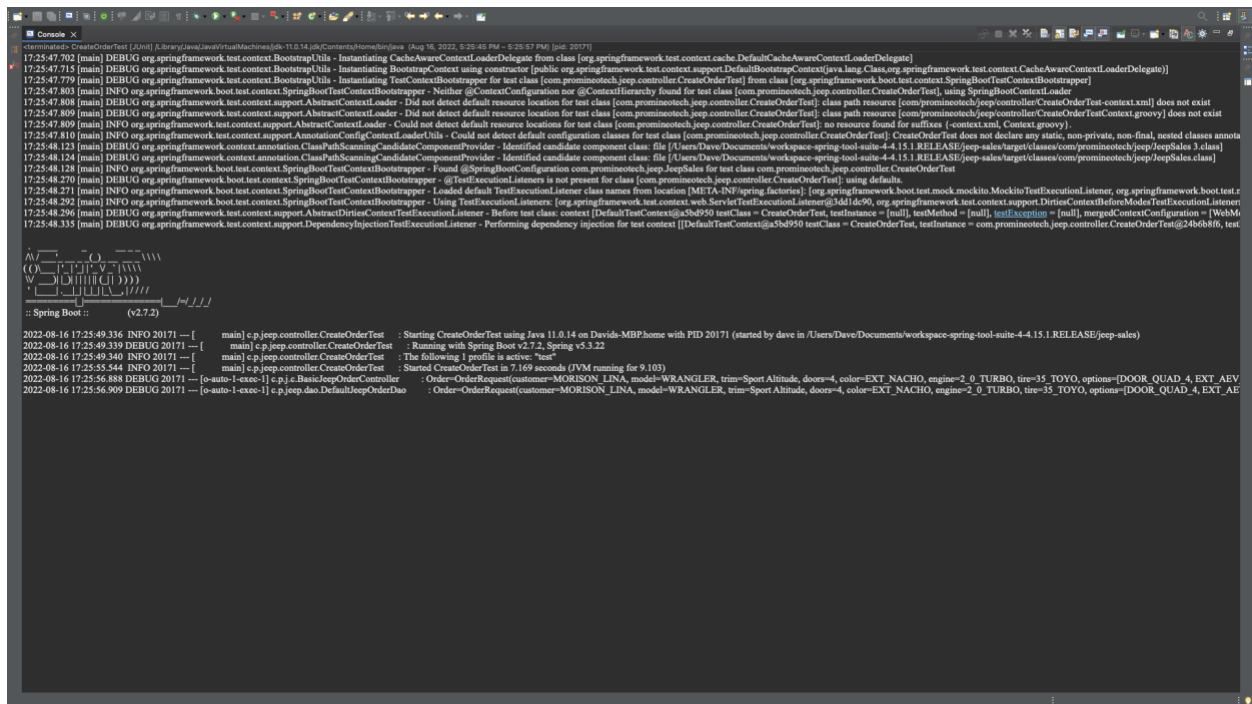
14) In DefaultJeepOrderService.java, work with the method createOrder.

- Add the `@Transactional` annotation to the `createOrder` method.
- 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.
- 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. 



- 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();
```

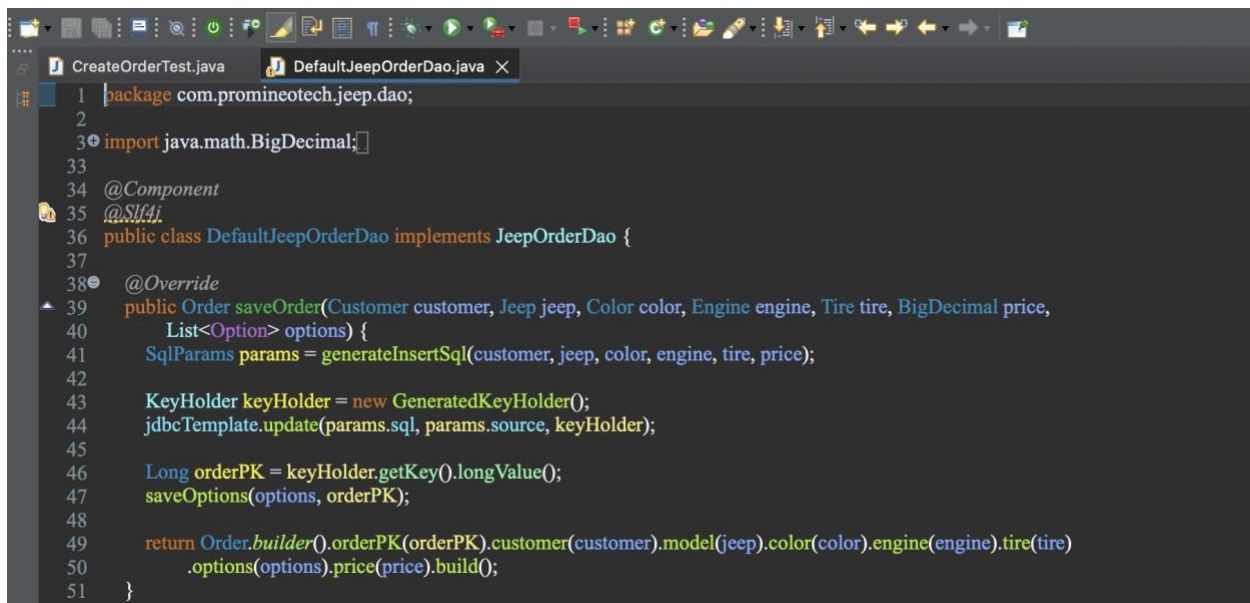
Be sure to extract the order primary key from the KeyHolder object into a variable of type Long named orderPK.

- 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. 



```
1 package com.promineotech.jeeo.dao;
2
3 import java.math.BigDecimal;
4
5 @Component
6 @Sql4j
7 public class DefaultJeepOrderDao implements JeepOrderDao {
8
9     @Override
10    public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price,
11        List<Option> options) {
12        SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
13
14        KeyHolder keyHolder = new GeneratedKeyHolder();
15        jdbcTemplate.update(params.sql, params.source, keyHolder);
16
17        Long orderPK = keyHolder.getKey().longValue();
18        saveOptions(options, orderPK);
19
20        return Order.builder().orderPK(orderPK).customer(customer).model(jeeo).color(color).engine(engine).tire(tire)
21            .options(options).price(price).build();
22    }
23 }
```

- 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. 

Package Explorer

JUnit

Finished after 9.192 seconds

Runs: 1/1

Errors: 0

Failures: 0

> CreateOrderTest [Runner: JUnit 5] (1.430 s)

Failure Trace

Screenshots of Code:

Screenshots of Running Application:

URL to GitHub Repository: