


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
 - 'EXT_DIAMOND_BLACK', 'Diamond Black Crystal Clear Coat', 245.00, 1
 - ii) Customer
 - 'ATTAWAY_HECKTOR', 'Hektor', 'Attaway', '755.223.5969'
 - iii) Engine
 - '3_0_DIESEL', 3.0, '3.0L V6 Turbo Diesel Engine with Start Stop', 'DIESEL', 22, 32, 1, 'The available 3.0L EcoDiesel V6 engine will offer up to 260 horsepower and 442 lb-ft of torque for impressive performance and outstanding low-end torque', 4500.00
 - iv) Model
 - 'GRAND_CHEROKEE', 'Limited', 4, 18, 40420.00
 - v) tire(s)
 - '265_MICHELIN', '265/60R18 BSW All-Season LRR Tires', 'Michelin', 0.00, 60000
- 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!
- 'EXT_MOPAR_KEYLESS', 'EXTERIOR', 'Mopar', 'Remote-Proximity Keyless-Entry', 645.00
 - 'INT_MOPAR_GRAB', 'INTERIOR', 'Mopar', 'Grab Handle Kit', 80.00
- 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.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. 
- 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. 
- 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. 

- 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`.

- 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. 
- 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. 

Screenshots of Code:

2c	<pre>70 71● String createOrderBody() { 72 return "{\n" 73 + " \"customer\": \"ATTAWAY_HECKTOR\", \n" 74 + " \"model\": \"GRAND_CHEROKEE\", \n" 75 + " \"trim\": \"Limited\", \n" 76 + " \"doors\": 4, \n" 77 + " \"color\": \"EXT_DIAMOND_BLACK\", \n" 78 + " \"engine\": \"3_0_DIESEL\", \n" 79 + " \"tire\": \"265_MICHELIN\", \n" 80 + " \"options\": [\n" 81 + " \"EXT_MOPAR_KEYLESS\", \n" 82 + " \"INT_MOPAR_GRAB\", \n" 83 + " \"EXT_WARN_WINCH\" \n" 84 + "] \n" 85 + "}"; 86 }</pre>
----	---

2k

The screenshot shows an IDE with a Java test class named `CreateOrderTest`. The test class is located in the package `com.promineotech.jeepp.controller`. It imports various Spring and JUnit dependencies. The test class has a method `testCreateOrderReturnsSuccess201` which is annotated with `@Test`. The test method is failing with an `AssertionFailedError`. The failure trace shows that the test expected a `201 CREATED` status but received a `404 NOT_FOUND` status. The test method is annotated with `@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)` and `@ActiveProfiles("test")`. The test method is annotated with `@Sql(scripts = {"classpath:flyway/migrations/V1.0__Jeep.Schema.sql", "classpath:flyway/migrations/V1.1__Jeep.Data.sql"})` and `config = @SqlConfig(encoding = "utf-8")`. The test method is annotated with `@LocalServerPort` and `@Autowired`. The test method is annotated with `@Test`. The test method is annotated with `void testCreateOrderReturnsSuccess201()`. The test method is annotated with `String uri = String.format("http://localhost:%d/orders", serverPort);`. The test method is annotated with `String body = createOrderBody();`. The test method is annotated with `HttpHeaders headers = new HttpHeaders();`. The test method is annotated with `headers.setContentType(MediaType.APPLICATION_JSON);`. The test method is annotated with `HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);`. The test method is annotated with `ResponseEntity<Order> response = restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, Order.class);`. The test method is annotated with `assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);`. The test method is annotated with `assertThat(response.getBody()).isNotNull();`. The test method is annotated with `Order order = response.getBody();`. The test method is annotated with `assertThat(order.getCustomer().getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");`. The test method is annotated with `assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GRAND_CHEROKEE);`. The test method is annotated with `assertThat(order.getModel().getTrimLevel()).isEqualTo("Limited");`. The test method is annotated with `assertThat(order.getModel().getNumDoors()).isEqualTo(4);`. The test method is annotated with `assertThat(order.getColor().getColorId()).isEqualTo("EXT_DIAMOND_BLACK");`. The test method is annotated with `assertThat(order.getEngine().getEngineId()).isEqualTo("3.0_DIESEL");`. The test method is annotated with `assertThat(order.getTire().getTireId()).isEqualTo("265_MICHELIN");`. The test method is annotated with `assertThat(order.getOptions()).hasSize(3);`. The test method is annotated with `String createOrderBody() {`. The test method is annotated with `return "{`. The test method is annotated with `"customer": "ATTAWAY_HECKTOR",`. The test method is annotated with `"model": "GRAND_CHEROKEE",`. The test method is annotated with `"trim": "Limited",`. The test method is annotated with `}`. The test method is annotated with `}`. The test method is annotated with `String createOrderBody() {`. The test method is annotated with `return "{`. The test method is annotated with `"customer": "ATTAWAY_HECKTOR",`. The test method is annotated with `"model": "GRAND_CHEROKEE",`. The test method is annotated with `"trim": "Limited",`. The test method is annotated with `}`. The test method is annotated with `}`.

```
1 package com.promineotech.jeepp.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import org.junit.jupiter.api.Test;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
8 import org.springframework.boot.test.web.client.TestRestTemplate;
9 import org.springframework.boot.web.server.LocalServerPort;
10 import org.springframework.http.HttpEntity;
11 import org.springframework.http.HttpHeaders;
12 import org.springframework.http.HttpMethod;
13 import org.springframework.http.HttpStatus;
14 import org.springframework.http.MediaType;
15 import org.springframework.http.ResponseEntity;
16 import org.springframework.test.context.ActiveProfiles;
17 import org.springframework.test.context.jdbc.Sql;
18 import org.springframework.test.context.jdbc.SqlConfig;
19 import com.promineotech.jeepp.entity.JeepModel;
20 import com.promineotech.jeepp.entity.Order;
21
22 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
23 @ActiveProfiles("test")
24 @Sql(scripts = {
25     "classpath:flyway/migrations/V1.0__Jeep.Schema.sql",
26     "classpath:flyway/migrations/V1.1__Jeep.Data.sql"),
27     config = @SqlConfig(encoding = "utf-8"))
28
29 public class CreateOrderTest {
30
31     @LocalServerPort
32     private int serverPort;
33
34     @Autowired
35     private TestRestTemplate restTemplate;
36
37     @Test
38     void testCreateOrderReturnsSuccess201() {
39         //Given
40         String uri = String.format("http://localhost:%d/orders", serverPort);
41         String body = createOrderBody();
42         HttpHeaders headers = new HttpHeaders();
43         headers.setContentType(MediaType.APPLICATION_JSON);
44         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
45
46         //When
47         ResponseEntity<Order> response = restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, Order.class);
48
49         //Then
50         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
51         assertThat(response.getBody()).isNotNull();
52
53         Order order = response.getBody();
54         assertThat(order.getCustomer().getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");
55         assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.GRAND_CHEROKEE);
56         assertThat(order.getModel().getTrimLevel()).isEqualTo("Limited");
57         assertThat(order.getModel().getNumDoors()).isEqualTo(4);
58         assertThat(order.getColor().getColorId()).isEqualTo("EXT_DIAMOND_BLACK");
59         assertThat(order.getEngine().getEngineId()).isEqualTo("3.0_DIESEL");
60         assertThat(order.getTire().getTireId()).isEqualTo("265_MICHELIN");
61         assertThat(order.getOptions()).hasSize(3);
62
63         //And
64
65     }
66
67     String createOrderBody() {
68         return "{
69             \"customer\": \"ATTAWAY_HECKTOR\",
70             \"model\": \"GRAND_CHEROKEE\",
71             \"trim\": \"Limited\",
72         }
73     }
```

Failure Trace

```
org.opentest4j.AssertionFailedError:
expected: 201 CREATED
but was: 404 NOT_FOUND
at java.base/java.lang.reflect.Constructor.newInstance()
at com.promineotech.jeepp.controller.CreateOrderTest.testCreateOrderReturnsSuccess201()
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541)
```

Boot Dashboard

Type tags, projects, or working set names to match

local

3c

```
JeepSales.java FetchJeepTe... JeepSalesDao... GlobalErrorH... CreateOrderT... JeepOrderCo... x JeepSalesCo... »  
1 package com.promineotech.jeepp.controller;  
2  
3 import org.springframework.http.HttpStatus;  
4 import org.springframework.web.bind.annotation.PostMapping;  
5 import org.springframework.web.bind.annotation.RequestBody;  
6 import org.springframework.web.bind.annotation.RequestMapping;  
7 import org.springframework.web.bind.annotation.ResponseStatus;  
8 import com.promineotech.jeepp.entity.Order;  
9 import com.promineotech.jeepp.entity.OrderRequest;  
10 import io.swagger.v3.oas.annotations.Operation;  
11 import io.swagger.v3.oas.annotations.Parameter;  
12 import io.swagger.v3.oas.annotations.media.Content;  
13 import io.swagger.v3.oas.annotations.media.Schema;  
14 import io.swagger.v3.oas.annotations.responses.ApiResponse;  
15  
16 @RequestMapping("/orders")  
17 public interface JeepOrderController {  
18  
19     @Operation(  
20         summary = "Create an Order for a Jeep",  
21         description = "Returns the created Jeep",  
22         responses = {  
23             @ApiResponse(  
24                 responseCode = "201",  
25                 description = "The created Jeep is returned",  
26                 content = @Content(  
27                     mediaType = "application/json",  
28                     schema = @Schema(implementation = Order.class))),  
29             @ApiResponse(  
30                 responseCode = "400",  
31                 description = "The request parameters are invalid",  
32                 content = @Content(mediaType = "application/json")),  
33             @ApiResponse(  
34                 responseCode = "404",  
35                 description = "A Jeep component was not found with the input criteria",  
36                 content = @Content(mediaType = "application/json")),  
37             @ApiResponse(  
38                 responseCode = "500",  
39                 description = "An unplanned error occurred",  
40                 content = @Content(mediaType = "application/json"))  
41         },  
42         parameters = {  
43             @Parameter(  
44                 name = "orderRequest",  
45                 required = true,  
46                 description = "The order as JSON")  
47         }  
48     )  
49     @PostMapping  
50     @ResponseStatus(code = HttpStatus.CREATED)  
51     Order createOrder(@RequestBody OrderRequest orderRequest);  
52  
53 }  
54
```

The image shows a screenshot of an IDE, likely IntelliJ IDEA, with a Java project. The main window displays the source code of a Spring Boot application. The code is organized into three main classes:
1. **Main Class:** A class that extends `SpringBootServletInitializer` and implements `configure` method to set up the web application.
2. **Configuration Class:** A class that implements `WebMvcConfigurer` and `HandlerMethodArgumentResolver` interfaces. It defines a `configure` method for `WebMvcConfigurer` and a `resolveArgument` method for `HandlerMethodArgumentResolver`.
3. **Service Class:** A class that implements `Service` interface and defines a `get` method.
The IDE also shows a terminal window at the bottom with the command `mvn clean install` and its output. The output shows the successful execution of the command and the generation of the application.
The IDE interface includes a project explorer on the left, a main editor window, and a terminal window at the bottom. The code is written in Java and uses standard annotations like `@SpringBootApplication`, `@Configuration`, `@Component`, and `@Autowired`.

7e

```
1 package com.promineotech.jeepp.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
```

[illegible]

```
15a
```

```
1 package com.promineotech.jeeep.service;
2
3 import java.math.BigDecimal;
4
5 @Service
6 @Slf4j
7 public class DefaultJeepOrderService implements JeepOrderService {
8
9     @Autowired
10    private JeepOrderDao jeepOrderDao;
11
12    @Transactional
13    @Override
14    public Order createOrder(OrderRequest orderRequest) {
15        Log.debug("Order={}", orderRequest);
16        Customer customer = getCustomer(orderRequest);
17        Jeep model = getModel(orderRequest);
18        Color color = getColor(orderRequest);
19        Engine engine = getEngine(orderRequest);
20        Tire tire = getTire(orderRequest);
21        List<Option> options = getOption(orderRequest);
22        BigDecimal price = model.getBasePrice().add(color.getPrice().add(engine.getPrice().add(tire.getPrice())));
23        for (Option each : options) {
24            price.add(each.getPrice());
25        }
26        return jeepOrderDao.saveOrder(customer, model, color, engine, tire, price, options);
27    }
28 }
```

15b.v

```
323     .modelPK(rs.getLong("model_pk"))
324     .numDoors(rs.getInt("num_doors"))
325     .trimLevel(rs.getString("trim_level"))
326     .wheelSize(rs.getInt("wheel_size"))
327     .build();
328     // @formatter:on
329 }
330 }
331
332 /**
333  *
334  * @author Promineo
335  *
336  */
337 class CustomerResultSetExtractor implements ResultSetExtractor<Customer> {
338     @Override
339     public Customer extractData(ResultSet rs) throws SQLException {
340         rs.next();
341
342         // @formatter:off
343         return Customer.builder()
344             .customerId(rs.getString("customer_id"))
345             .customerPK(rs.getLong("customer_pk"))
346             .firstName(rs.getString("first_name"))
347             .lastName(rs.getString("last_name"))
348             .phone(rs.getString("phone"))
349             .build();
350         // @formatter:on
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     SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
363     KeyHolder keyHolder = new GeneratedKeyHolder();
364     jdbcTemplate.update(params.sql, params.source, keyHolder);
365
366     Long orderPK = keyHolder.getKey().longValue();
367     saveOptions(options, orderPK);
368     return Order.builder()
369         .orderPK(orderPK)
370         .customer(customer)
371         .model(jeep)
372         .color(color)
373         .engine(engine)
374         .tire(tire)
375         .options(options)
376         .price(price)
377         .build();
378 }
379
380 private void saveOptions(List<Option> options, Long orderPK) {
381     for (Option each : options) {
382         SqlParams params = generateInsertSql(each, orderPK);
383         jdbcTemplate.update(params.sql, params.source);
384     }
385 }
386
387 }
388
389 }
390
```

Problems Javadoc Declaration Console X

