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.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 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.

```
1  package com.promineotech.jeep.controller.support;
2
3  public class CreateOrderTestSupport extends BaseTest {
4
5⊖   protected String createOrderBody() {
5        // @formatter:off
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       // @formatter:on
25   }
26 }
27
```

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.jeep.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.

```java
@Test
void testCreateOrderReturnsSuccess201() {

    // Given: an order as JSON
    String body = createOrderBody();
    String uri = getBaseUriForOrders();
    int numRowsOrders = JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders");
    int numRowsOptions = JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options");
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

    // When: The order is sent
    ResponseEntity<Order> response =
        getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity, Order.class);

    // Then: a 201 status is returned
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);

    // And: the returned order is correct
    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);
    assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders"))
        .isEqualTo(numRowsOrders + 1);
    assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options"))
        .isEqualTo(numRowsOptions + 6);
}
```

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.

```java
 1  package com.promineotech.jeep.controller;
 2
 3⊕ import javax.validation.Valid;
20
21  @Validated
22  @RequestMapping("/orders")
23  @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"),
24      servers = {@Server(url = "http://localhost:8080", description = "Local server.")})
25
26  public interface JeepOrderController {
27
28    //@formatter:off
29⊝   @Operation(
30        summary = "Create an order for a jeep",
31        description = "Returns the created Jeep",
32          //@formatter:off
33        responses = {
34            @ApiResponse(
35                responseCode = "201",
36                description = "The created Jeep is returned",
37                content = @Content(
38                    mediaType = "application/json",
39                    schema = @Schema(implementation = Order.class))),
40            @ApiResponse(
41                responseCode = "400",
42                description = "The request parameters are invalid",
43                content = @Content(
44                    mediaType = "application/json")),
45            @ApiResponse(
46                responseCode = "404",
47                description = "A Jeep component was not found with the input criteria",
48                content = @Content(
49                    mediaType = "application/json")),
50            @ApiResponse(
51                responseCode = "500",
52                description = "An unplanned error occurred.",
53                content = @Content(
54                    mediaType = "application/json"))
55        },
56        parameters = {
57            @Parameter(
58                name = "orderRequest",
59                required = true,
60                description = "The order is JSON"),
61
62        }
63    //@formatter:on
64    )
65
66
67    @PostMapping
68    @ResponseStatus(code = HttpStatus.CREATED)
69    Order createOrder(@Valid @RequestBody OrderRequest orderRequest);
70
71
72  }
73
```

| Writable | Smart Insert | 1:1:0 | ⋮ |

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.

Console ✕ | Problems | Debug Shell

```
<terminated> CreateOrderTest [JUnit] C:\Program Files\Java\jdk-17.0.3\bin\javaw.exe  (Aug 21, 2022, 6:26:50 PM – 6:27:00 PM) [
springframework.boot.test.autoconfigure.actuate.metrics.MetricsExportContextCust
omizerFactory$DisableMetricExportContextCustomizer@27ff5d15, org.springframework
.boot.test.autoconfigure.properties.PropertyMappingContextCustomizer@0, org.spri
ngframework.boot.test.autoconfigure.web.servlet.WebDriverContextCustomizerFactor
y$Customizer@15aab8c6, org.springframework.boot.test.context.SpringBootTestArgs@
1, org.springframework.boot.test.context.SpringBootTestWebEnvironment@4fe3c938],
 resourceBasePath = 'src/main/webapp', contextLoader = 'org.springframework.boot
.test.context.SpringBootContextLoader', parent = [null]], attributes = map['org.
springframework.test.context.web.ServletTestExecutionListener.activateListener'
-> false, 'org.springframework.test.context.event.ApplicationEventsTestExecution
Listener.recordApplicationEvents' -> false]]].


  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.2)

2022-08-21 18:26:52.468  INFO 21472 --- [           main] c.p.jeep.controller.Cr
eateOrderTest         : Starting CreateOrderTest using Java 17.0.3 on LAPTOP-1N0U2V
3V with PID 21472 (started by Jared in C:\Users\Jared\OneDrive\Desktop\Promineo\
Eclipse\jeep-sales)
2022-08-21 18:26:52.468 DEBUG 21472 --- [           main] c.p.jeep.controller.Cr
eateOrderTest         : Running with Spring Boot v2.7.2, Spring v5.3.22
2022-08-21 18:26:52.469  INFO 21472 --- [           main] c.p.jeep.controller.Cr
eateOrderTest         : The following 1 profile is active: "test"
2022-08-21 18:26:59.202  INFO 21472 --- [           main] c.p.jeep.controller.Cr
eateOrderTest         : Started CreateOrderTest in 7.141 seconds (JVM running for 8
.502)
2022-08-21 18:27:00.528 DEBUG 21472 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrd
erController           : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, t
rim=Sport Altitude, doors=4, color=EXT_NACHO, engine=2_0_TURBO, tire=35_TOYO, op
tions=[DOOR_QUAD_4, EXT_AEV_LIFT, EXT_WARN_WINCH, EXT_WARN_BUMPER_FRONT, EXT_WAR
N_BUMPER_REAR, EXT_ARB_COMPRESSOR])
```

```java
@Test
void testCreateOrderReturnsSuccess201() {

    // Given: an order as JSON
    String body = createOrderBody();
    String uri = getBaseUriForOrders();
    int numRowsOrders = JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders");
    int numRowsOptions = JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options");
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

    // When: The order is sent
    ResponseEntity<Order> response =
        getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity, Order.class);

    // Then: a 201 status is returned
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);

    // And: the returned order is correct
    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);
    assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders"))
        .isEqualTo(numRowsOrders + 1);
    assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options"))
        .isEqualTo(numRowsOptions + 6);
  }

}
```

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. 🖥️

```
2   @Data
3   public class OrderRequest {
4⊖      @NotNull
5       @Length(max = 30)
6       @Pattern(regexp = "[\\w\\s]*")
7       private String customer;
8
9⊖      @NotNull
0       private JeepModel model;
1
2⊖      @NotNull
3       @Length(max = 30)
4       @Pattern(regexp = "[\\w\\s]*")
5       private String trim;
6
7⊖      @Positive
8       @Min(2)
9       @Max(4)
0       private int doors;
1
2⊖      @NotNull
3       @Length(max = 30)
4       @Pattern(regexp = "[\\w\\s]*")
5       private String color;
6
7⊖      @NotNull
8       @Length(max = 30)
9       @Pattern(regexp = "[\\w\\s]*")
0       private String engine;
1
2⊖      @NotNull
3       @Length(max = 30)
4       @Pattern(regexp = "[\\w\\s]*")
5       private String tire;
6
7       private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String> options;
8   }
9
```

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.

```java
@Transactional
@Override
public Order createOrder(OrderRequest orderRequest) {
    Customer customer = getCustomer(orderRequest);
    Jeep jeep = getModel(orderRequest);
    Color color = getColor(orderRequest);
    Engine engine = getEngine(orderRequest);
    Tire tire = getTire(orderRequest);
    List<Option> options = getOption(orderRequest);
    BigDecimal price = jeep.getBasePrice().add(color.getPrice()).add(engine.getPrice()).add(tire.getPrice());

    for(Option option : options) {
        price = price.add(option.getPrice());
    }

    return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
}
```

    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. 🖥️

```java
    @Override
    public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
        BigDecimal price, List<Option> options) {
        SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);

        KeyHolder keyHolder = new GeneratedKeyHolder();
        jdbcTemplate.update(params.sql, params.source, keyHolder);

        Long orderPK = keyHolder.getKey().longValue();
        saveOptions(options, orderPK);

        //@formatter:off
        return Order.builder()
            .orderPK(orderPK)
            .customer(customer)
            .model(jeep)
            .color(color)
            .engine(engine)
            .tire(tire)
            .options(options)
            .price(price)
            .build();
        //@formatter:on
    }

    private void saveOptions(List<Option> options, Long orderPK) {
        for (Option option : options) {
            SqlParams params = generateInsertSql(option, orderPK);

            jdbcTemplate.update(params.sql, params.source);

        }
```

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. 🖥️

```java
@Test
void testCreateOrderReturnsSuccess201() {

    // Given: an order as JSON
    String body = createOrderBody();
    String uri = getBaseUriForOrders();
    int numRowsOrders = JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders");
    int numRowsOptions = JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options");
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

    // When: The order is sent
    ResponseEntity<Order> response =
        getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity, Order.class);

    // Then: a 201 status is returned
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);

    // And: the returned order is correct
    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);
    assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders"))
    .isEqualTo(numRowsOrders + 1);
    assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options"))
    .isEqualTo(numRowsOptions + 6);
    }

}
```
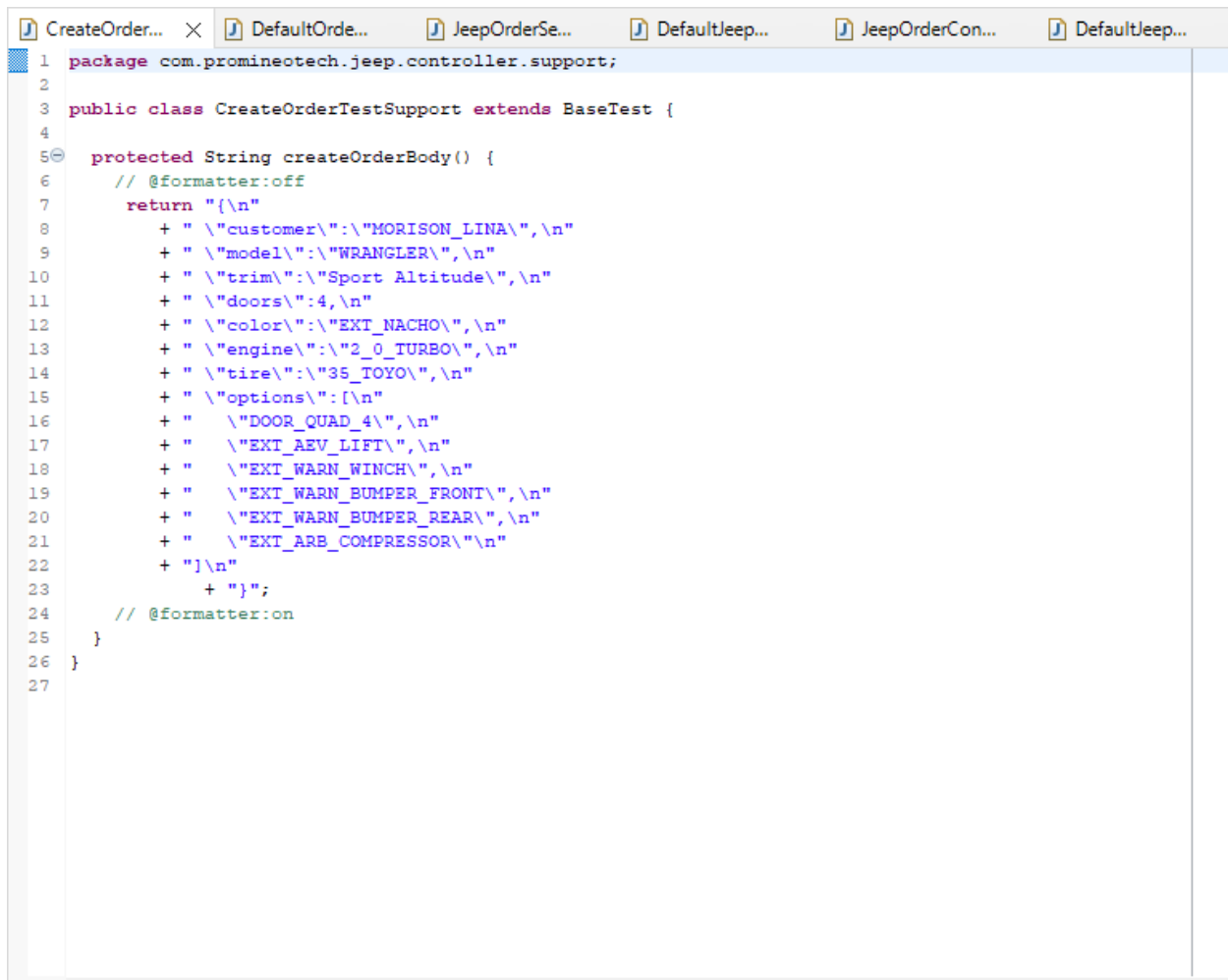
Screenshots of Code:

```java
package com.promineotech.jeep.controller.support;

public class CreateOrderTestSupport extends BaseTest {

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

```java
 1  package com.promineotech.jeep.controller;
 2
 3⊕ import static org.assertj.core.api.Assertions.assertThat;
22
23  @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
24  @ActiveProfiles("test")
25  @Sql(scripts = {"classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
26      "classpath:flyway/migrations/V1.1__Jeep_Data.sql"}, config = @SqlConfig(encoding = "utf-8"))
27  class CreateOrderTest extends CreateOrderTestSupport {
28
29⊖   @Autowired
30    private JdbcTemplate jdbcTemplate;
31
32⊖   @Test
33    void testCreateOrderReturnsSuccess201() {
34
35      // Given: an order as JSON
36      String body = createOrderBody();
37      String uri = getBaseUriForOrders();
38      int numRowsOrders = JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders");
39      int numRowsOptions = JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options");
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 =
46          getRestTemplate().exchange(uri, 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);
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      assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "orders"))
63      .isEqualTo(numRowsOrders + 1);
64      assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "order_options"))
65      .isEqualTo(numRowsOptions + 6);
66    }
67
68  }
69
```

Writable | Smart Insert | 53 : 34 : 2281

```java
  1  package com.promineotech.jeep.service;
  2
  3⊕ import java.math.BigDecimal;
 18
 19  @Service
 20  public class DefaultOrderService 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      BigDecimal price = jeep.getBasePrice().add(color.getPrice()).add(engine.getPrice()).add(tire.getPr
 35
 36      for(Option option : options) {
 37        price = price.add(option.getPrice());
 38      }
 39
 40      return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
 41    }
 42
 43⊖   /**
 44     *
 45     * @param orderRequest
 46     * @return
 47     */
 48⊖   private List<Option> getOption(OrderRequest orderRequest) {
 49      return jeepOrderDao.fetchOptions(orderRequest.getOptions());
 50    }
 51
 52⊖   /**
 53     *
```

Writable   Smart Insert   1:1:0

```java
53      *
54      * @param orderRequest
55      * @return
56      */
57     private Tire getTire(OrderRequest orderRequest) {
58       return jeepOrderDao.fetchTire(orderRequest.getTire())
59         .orElseThrow(() -> new NoSuchElementException(
60             "Tire with ID=" + orderRequest.getTire() + " was not found"));
61     }
62
63     /**
64      *
65      * @param orderRequest
66      * @return
67      */
68     private Engine getEngine(OrderRequest orderRequest) {
69       return jeepOrderDao.fetchEngine(orderRequest.getEngine())
70         .orElseThrow(() -> new NoSuchElementException(
71             "Engine with ID=" + orderRequest.getEngine() + " was not found"));
72     }
73
74     /**
75      *
76      * @param orderRequest
77      * @return
78      */
79     private Color getColor(OrderRequest orderRequest) {
80       return jeepOrderDao.fetchColor(orderRequest.getColor())
81         .orElseThrow(() -> new NoSuchElementException(
82             "Color with ID=" + orderRequest.getColor() + " was not found"));
83     }
84
85     /**
86      *
87      * @param orderRequest
88      * @return
89      */
90     private Jeep getModel(OrderRequest orderRequest) {
```

Tabs: JeepOrderService.java × DefaultJeepOrderController.java JeepOrderController.java DefaultJeepOrderDao.java JeepOrde

```java
 1  package com.promineotech.jeep.service;
 2
 3  import com.promineotech.jeep.entity.Order;
 4
 5
 6  public interface JeepOrderService {
 7
 8    Order createOrder(OrderRequest orderRequest);
 9
10  }
```

```java
 1  package com.promineotech.jeep.controller;
 2
 3⊕ import org.springframework.beans.factory.annotation.Autowired;▢
 9
10  @RestController
11  @Slf4j
12  public class DefaultJeepOrderController implements JeepOrderController {
13
14⊖   @Autowired
15    private JeepOrderService jeepOrderService;
16
17⊖   @Override
△18   public Order createOrder(OrderRequest orderRequest) {
19      log.debug("Order={}", orderRequest);
20      return jeepOrderService.createOrder(orderRequest);
21    }
22
23  }
24
```

Tabs: JeepOrderController.java ✕ | DefaultJeepOrderDao.java | JeepOrderDao.java

```java
 1  package com.promineotech.jeep.controller;
 2
 3⊕ import javax.validation.Valid;▢
20
21  @Validated
22  @RequestMapping("/orders")
23  @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"),
24      servers = {@Server(url = "http://localhost:8080", description = "Local server.")})
25
26  public interface JeepOrderController {
27
28    //@formatter:off
29⊖   @Operation(
30        summary = "Create an order for a jeep",
31        description = "Returns the created Jeep",
32          //@formatter:off
33        responses = {
34          @ApiResponse(
35              responseCode = "201",
36              description = "The created Jeep is returned",
37              content = @Content(
38                  mediaType = "application/json",
39                  schema = @Schema(implementation = Order.class))),
40          @ApiResponse(
41              responseCode = "400",
42              description = "The request parameters are invalid",
43              content = @Content(
44                  mediaType = "application/json")),
45          @ApiResponse(
46              responseCode = "404",
47              description = "A Jeep component was not found with the input criteria",
48              content = @Content(
49                  mediaType = "application/json")),
50          @ApiResponse(
51              responseCode = "500",
52              description = "An unplanned error occurred.",
53              content = @Content(
54                  mediaType = "application/json"))
55          }
```

```
55         },
56         parameters = {
57             @Parameter(
58                 name = "orderRequest",
59                 required = true,
60                 description = "The order is JSON"),
61
62         }
63     //@formatter:on
64   )
65
66
67     @PostMapping
68     @ResponseStatus(code = HttpStatus.CREATED)
69     Order createOrder(@Valid @RequestBody OrderRequest orderRequest);
70
71
72 }
73
```

Writable | Smart Insert | 1:1:0

DefaultJeepOrderDao.java ✕ | JeepOrderDao.java

```
 1  package com.promineotech.jeep.dao;
 2
 3⊕ import java.math.BigDecimal;⬚
29
30  @Component
31  public class DefaultJeepOrderDao implements JeepOrderDao {
32
33⊖    @Autowired
34     private NamedParameterJdbcTemplate jdbcTemplate;
35
36
37⊖    @Override
38     public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
39         BigDecimal price, List<Option> options) {
40       SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
41
42       KeyHolder keyHolder = new GeneratedKeyHolder();
43       jdbcTemplate.update(params.sql, params.source, keyHolder);
44
45       Long orderPK = keyHolder.getKey().longValue();
46       saveOptions(options, orderPK);
47
48       //@formatter:off
49       return Order.builder()
50           .orderPK(orderPK)
51           .customer(customer)
52           .model(jeep)
53           .color(color)
54           .engine(engine)
55           .tire(tire)
56           .options(options)
57           .price(price)
58           .build();
59       //@formatter:on
60     }
61
62⊖    private void saveOptions(List<Option> options, Long orderPK) {
63       for (Option option : options) {
64         SqlParams params = generateInsertSql(option, orderPK);
```

```
78      SqlParams params = new SqlParams();
79
80      // @formatter:off
81      params.sql = ""
82          + "INSERT INTO order_options ("
83          + "option_fk, order_fk"
84          + ") VALUES ("
85          + ":option_fk, :order_fk"
86          + ")";
87      // @formatter:on
88
89      params.source.addValue("option_fk", option.getOptionPK());
90      params.source.addValue("order_fk", orderPK);
91
92      return params;
93  }
94
95⊝  /**
96    *
97    * @param customer
98    * @param jeep
99    * @param color
```

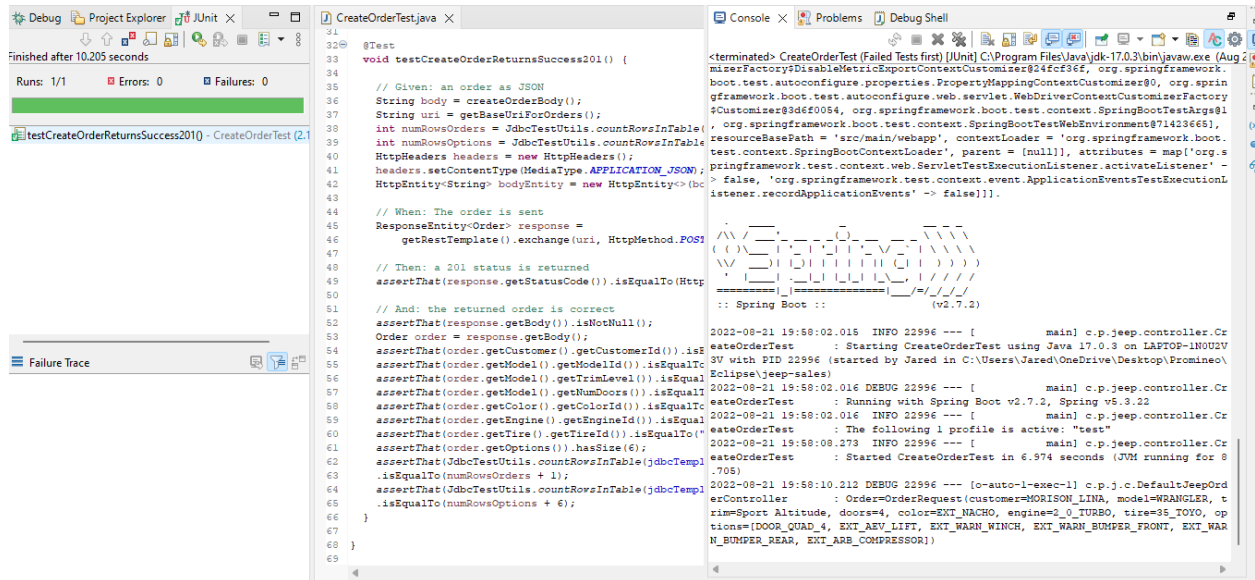| Writable	Smart Insert	1:1:0

JeepOrderDao.java ×

```
 1  package com.promineotech.jeep.dao;
 2
 3⊕ import java.math.BigDecimal;▯
14
15  public interface JeepOrderDao {
16      List<Option> fetchOptions(List<String> optionIds);
17      Optional<Customer> fetchCustomer(String customerId);
18      Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
19      Optional<Color> fetchColor(String colorId);
20      Optional<Engine> fetchEngine(String engineId);
21      Optional<Tire> fetchTire(String tireId);
22⊝     Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
23          BigDecimal price, List<Option> options);
24  }
25
```

Screenshots of Running Application:

URL to GitHub Repository:

https://github.com/JaredBears/JeepSales