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.

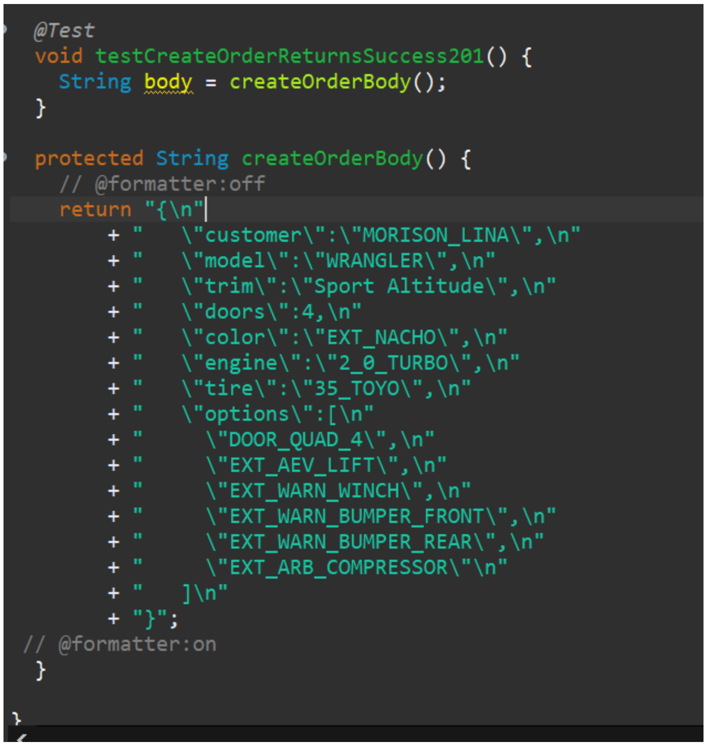
   b) Create a test method (annotated with `@Test`) named `testCreateOrderReturnsSuccess201`.

   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. 🖥



```
  @Test
  void testCreateOrderReturnsSuccess201() {
    String body = createOrderBody();
  }

  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
  }

}
```

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);
```

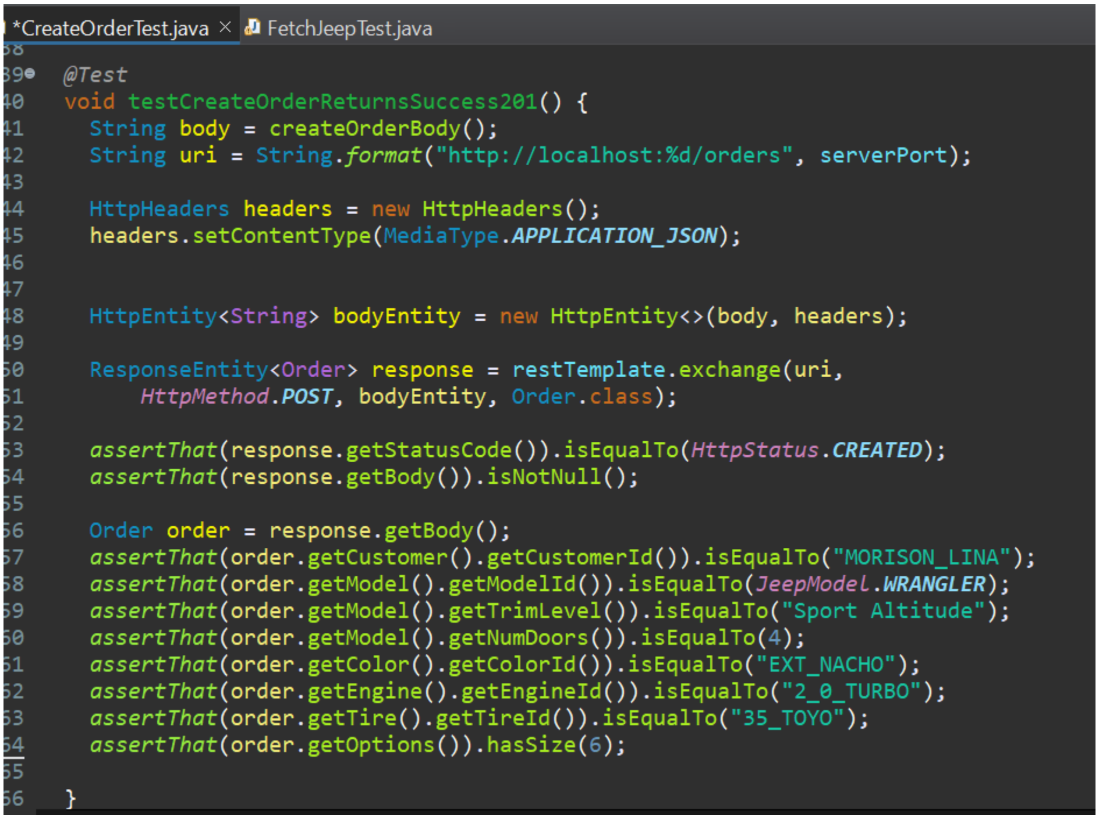
i) Send the request body and headers to the server. The Order class should have been copied earlier from the supplied resources. Ensure that you import `com.promineotech.jeep.entity.Order` and not some other Order class.

```
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.



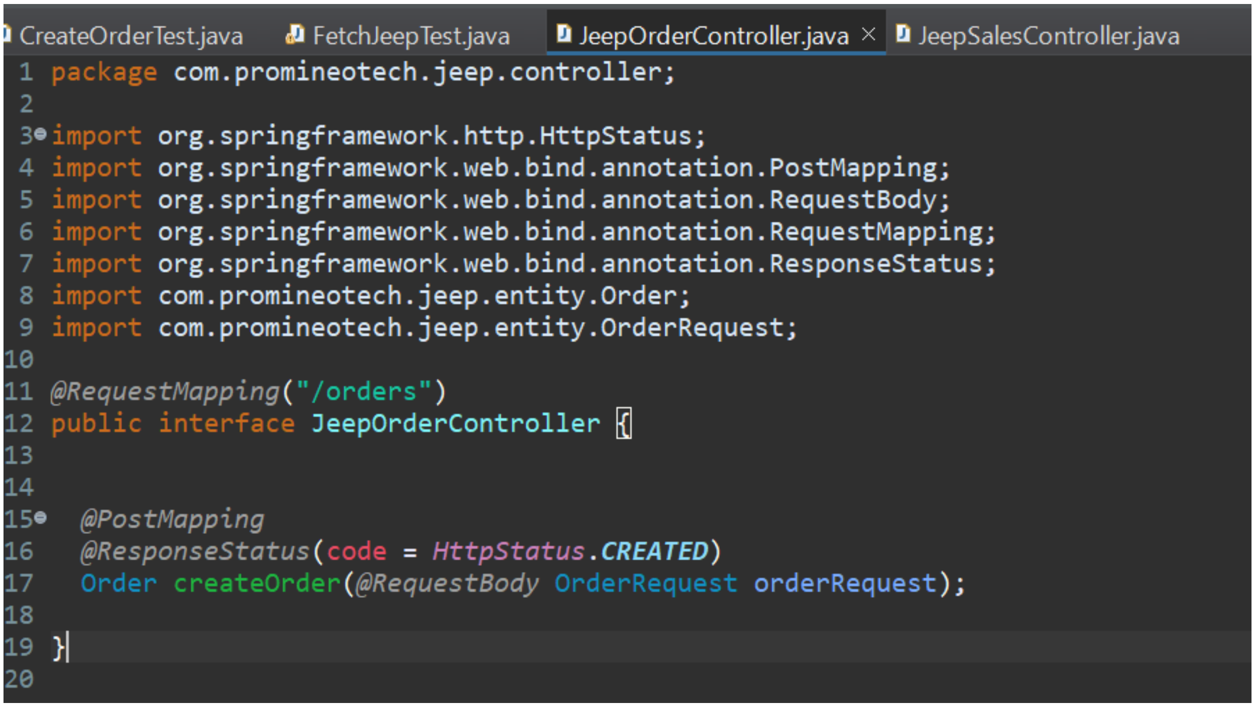
```
  @Test
  void testCreateOrderReturnsSuccess201() {
    String body = createOrderBody();
    String uri = String.format("http://localhost:%d/orders", serverPort);

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);


    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

    ResponseEntity<Order> response = restTemplate.exchange(uri,
        HttpMethod.POST, bodyEntity, Order.class);

    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);

  }
```

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.



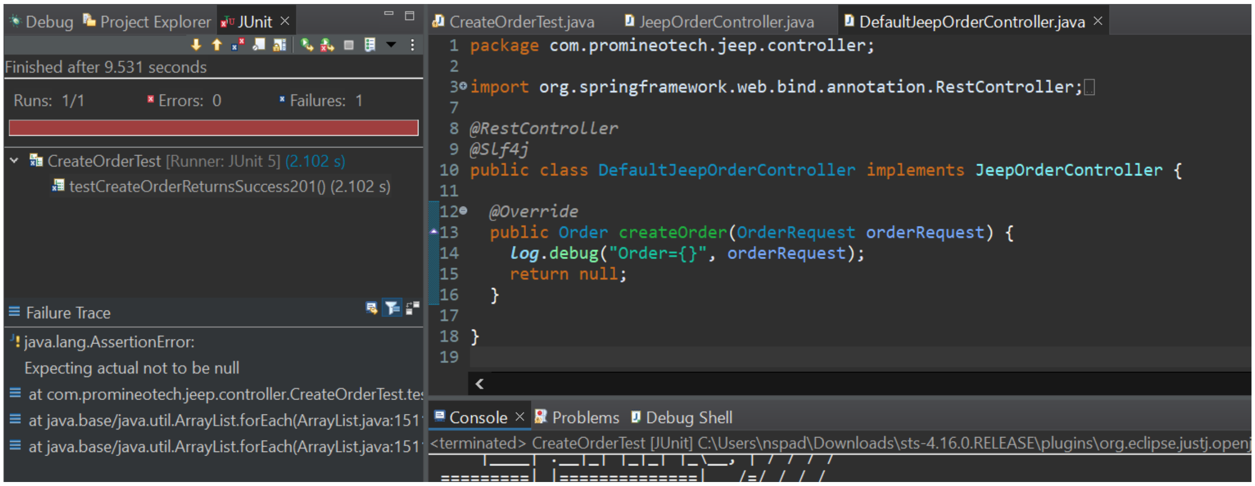
```
package com.promineotech.jeep.controller;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import com.promineotech.jeep.entity.Order;
import com.promineotech.jeep.entity.OrderRequest;

@RequestMapping("/orders")
public interface JeepOrderController {


  @PostMapping
  @ResponseStatus(code = HttpStatus.CREATED)
  Order createOrder(@RequestBody OrderRequest orderRequest);

}

```

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.



```
Finished after 9.531 seconds

Runs:  1/1        Errors:  0        Failures:  1

CreateOrderTest [Runner: JUnit 5] (2.102 s)
    testCreateOrderReturnsSuccess201() (2.102 s)

java.lang.AssertionError:
Expecting actual not to be null
at com.promineotech.jeep.controller.CreateOrderTest.tes
at java.base/java.util.ArrayList.forEach(ArrayList.java:151
at java.base/java.util.ArrayList.forEach(ArrayList.java:151
```

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

import org.springframework.web.bind.annotation.RestController;

@RestController
@Slf4j
public class DefaultJeepOrderController implements JeepOrderController {

  @Override
  public Order createOrder(OrderRequest orderRequest) {
    log.debug("Order={}", orderRequest);
    return null;
  }

}
```

```
<terminated> CreateOrderTest [JUnit] C:\Users\nspad\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justj.openj
      |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
```

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)`

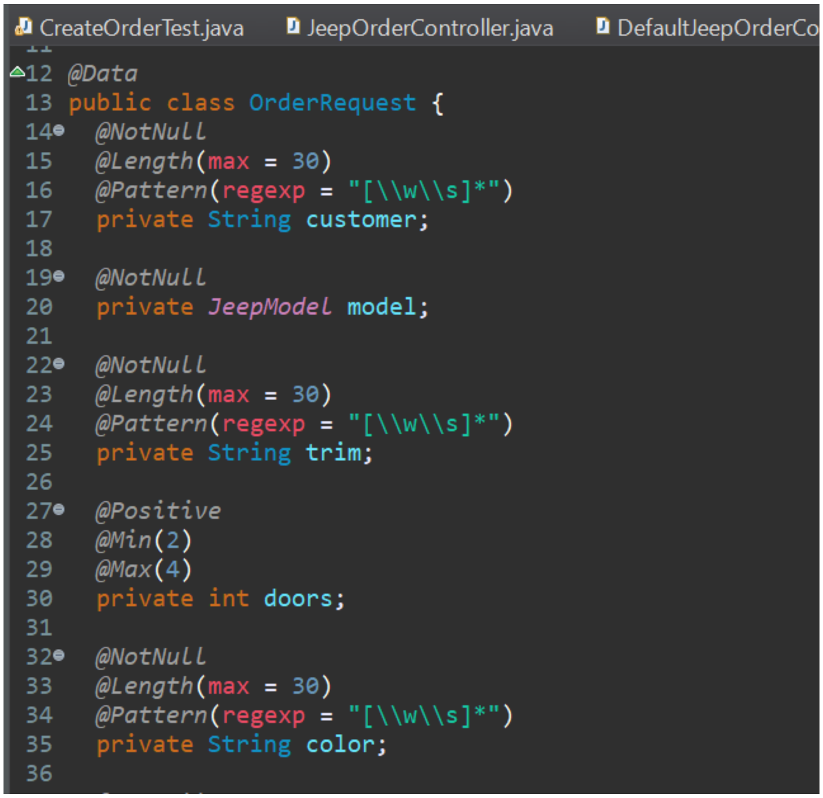
   c) Add `@NotNull` to the enum type.

   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:

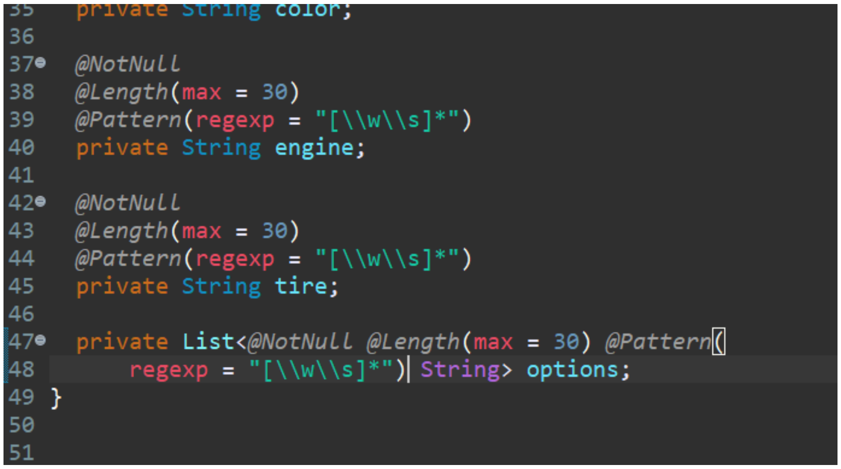
```java
private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*") String>
options;
```

   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. 🖥



```java
@Data
public class OrderRequest {
  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String customer;

  @NotNull
  private JeepModel model;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String trim;

  @Positive
  @Min(2)
  @Max(4)
  private int doors;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String color;

```



```java
  private String color;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String engine;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String tire;

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

8) In the `jeep.service` sub-package, create the empty (no methods yet) order service interface (named `JeepOrderService`) and implementation (named `DefaultJeepOrderService`).

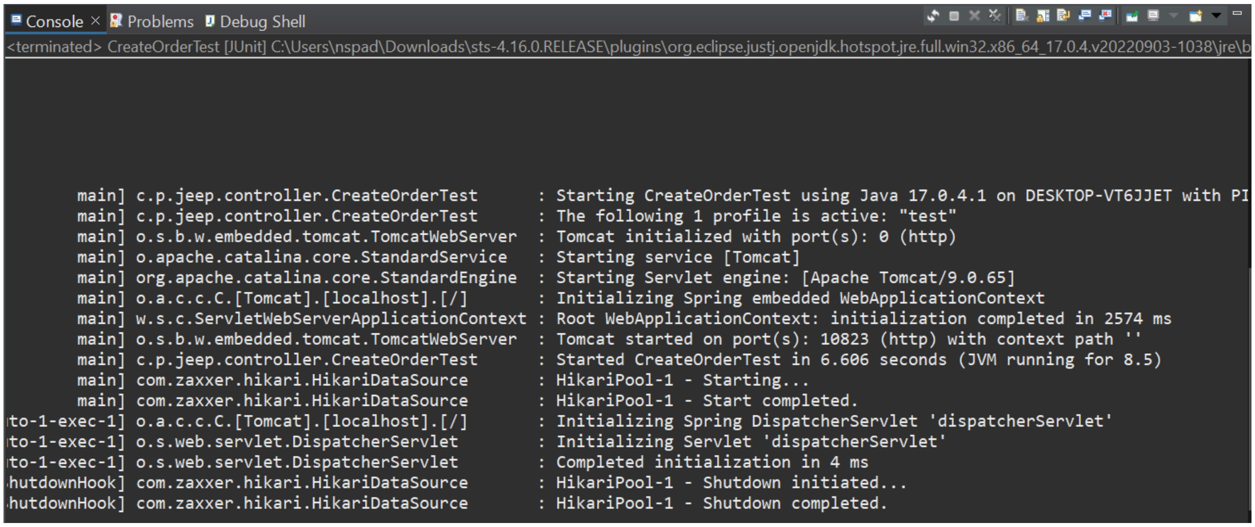
a) Inject the interface into the order controller implementation class.
b) Add the `@Service` annotation to the service implementation class.
c) Create the `createOrder` method in the interface and implementing service. The method signature should look like this:

```java
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).



```
<terminated> CreateOrderTest [JUnit] C:\Users\nspad\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\b
```

```
       main] c.p.jeep.controller.CreateOrderTest      : Starting CreateOrderTest using Java 17.0.4.1 on DESKTOP-VT6JJET with PI
       main] c.p.jeep.controller.CreateOrderTest      : The following 1 profile is active: "test"
       main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 0 (http)
       main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
       main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
       main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
       main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2574 ms
       main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 10823 (http) with context path ''
       main] c.p.jeep.controller.CreateOrderTest      : Started CreateOrderTest in 6.606 seconds (JVM running for 8.5)
       main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
       main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
to-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
to-1-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
to-1-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 4 ms
hutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
hutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

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.

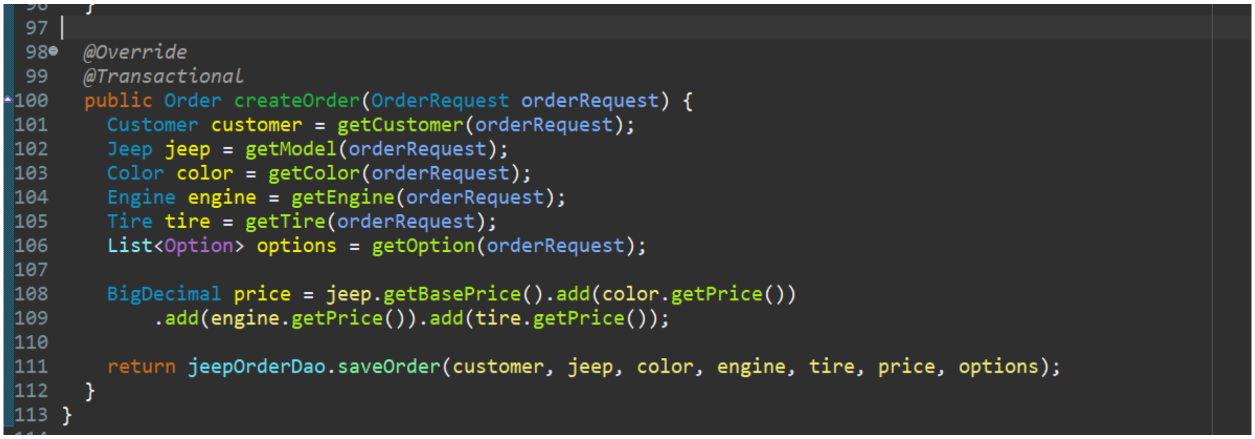
b) In the `createOrder` method call the copied methods: `getCustomer`, `getModel`, `getColor`, `getEngine`, `getTire` and `getOption`, assigning the return values of these methods to variables of the appropriate types.

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.



```
  }

  @Override
  @Transactional
  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());

    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`.

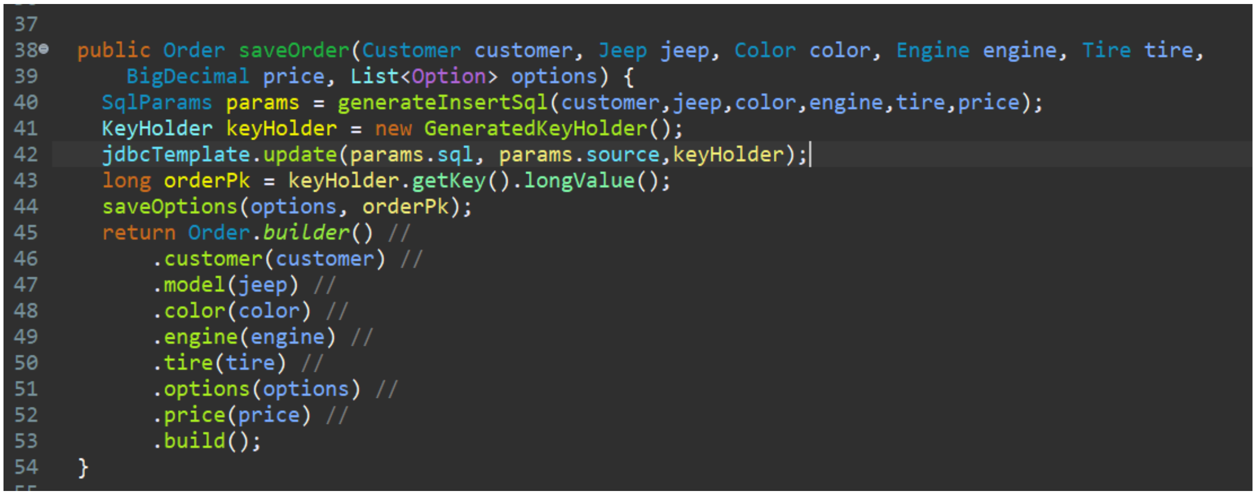
iii) Write a method named `saveOptions` as shown in the video. This method should have the following method signature:

```
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.

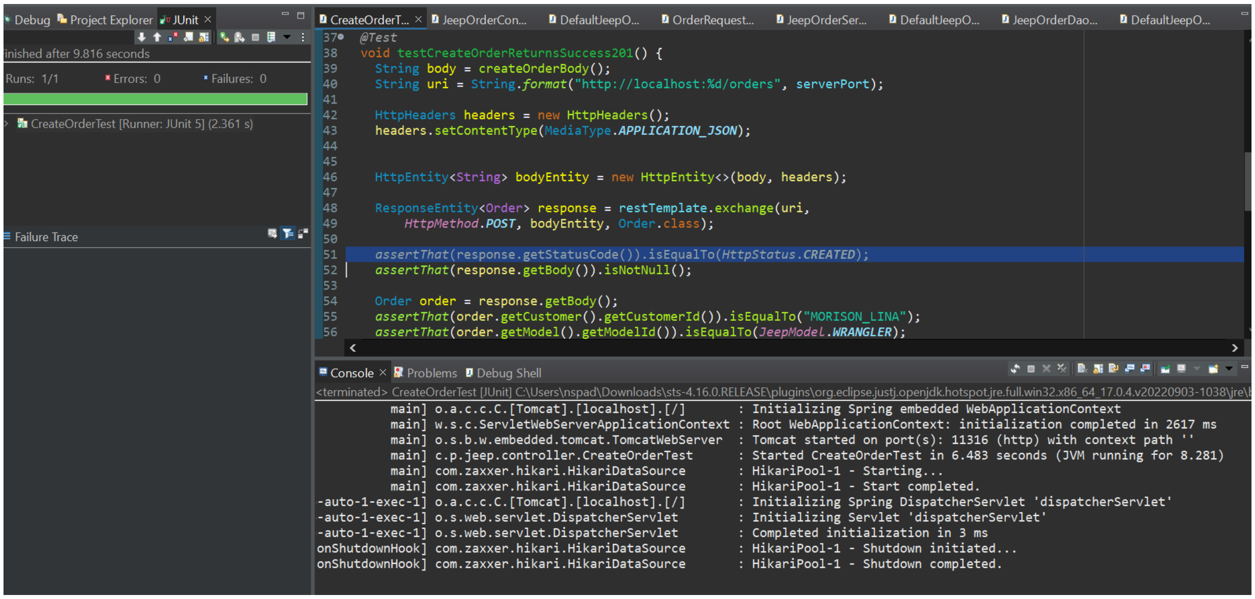
iv) In the `saveOrder` method in the DAO implementation, return an `Order` object using the `Order.builder`. The `Order` should include orderPK, customer, jeep (model), color, engine, tire, options and price.

v) Produce a screenshot of the saveOrder method.



```java
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);
  return Order.builder() //
      .customer(customer) //
      .model(jeep) //
      .color(color) //
      .engine(engine) //
      .tire(tire) //
      .options(options) //
      .price(price) //
      .build();
}
```

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.



inished after 9.816 seconds

Runs: 1/1   Errors: 0   Failures: 0

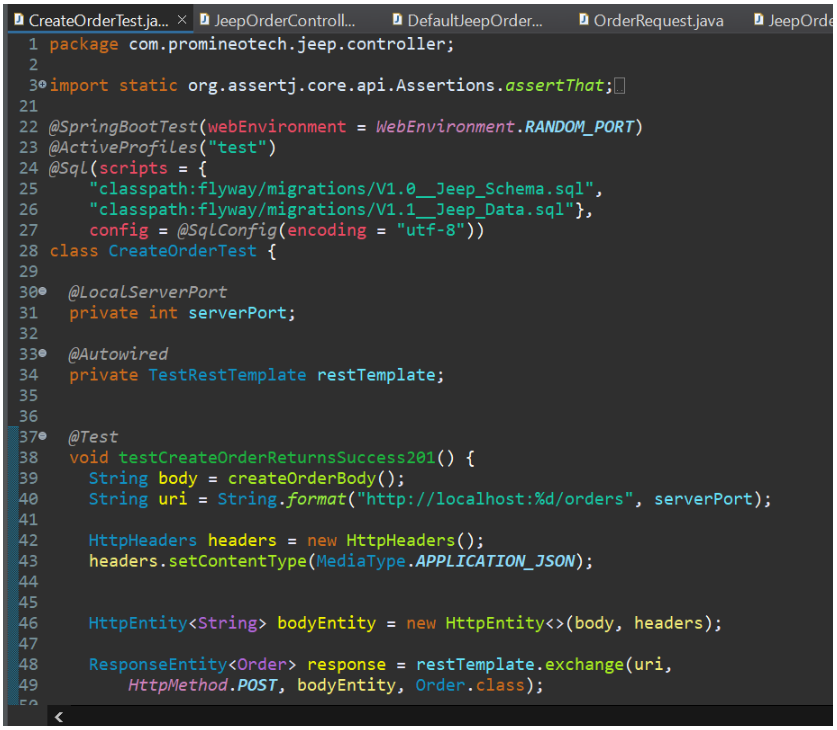
CreateOrderTest [Runner: JUnit 5] (2.361 s)

Failure Trace

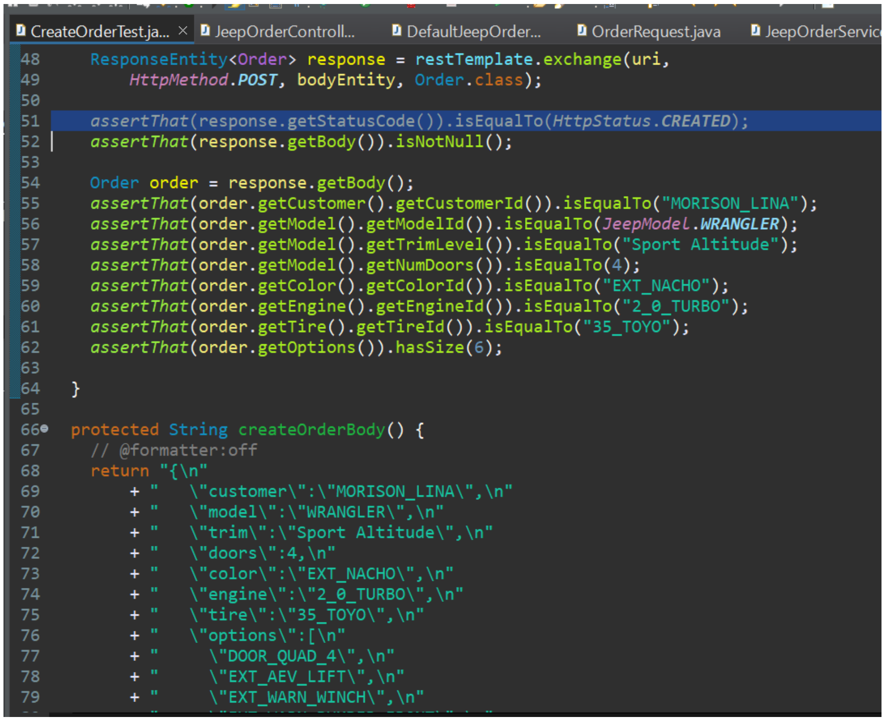
```java
@Test
void testCreateOrderReturnsSuccess201() {
  String body = createOrderBody();
  String uri = String.format("http://localhost:%d/orders", serverPort);

  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_JSON);


  HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

  ResponseEntity<Order> response = restTemplate.exchange(uri,
      HttpMethod.POST, bodyEntity, Order.class);

  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);
```

```
<terminated> CreateOrderTest [JUnit] C:\Users\nspad\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\
          main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
          main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2617 ms
          main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 11316 (http) with context path ''
          main] c.p.jeep.controller.CreateOrderTest      : Started CreateOrderTest in 6.483 seconds (JVM running for 8.281)
          main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
          main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 ms
onShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
onShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

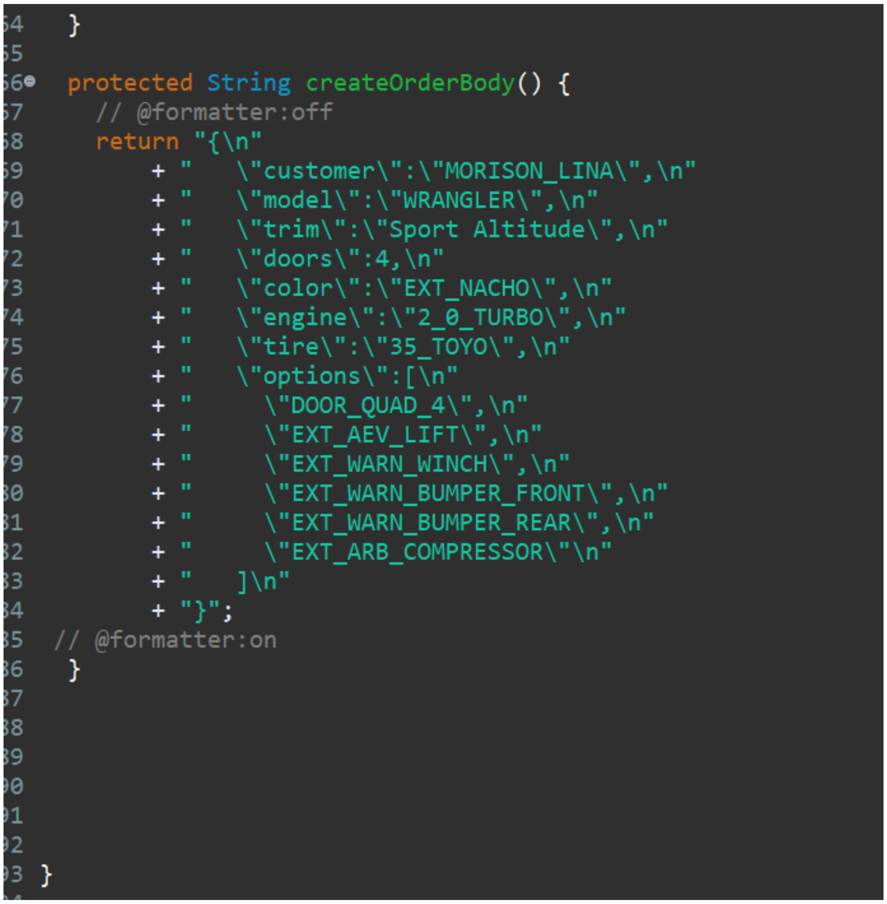
Screenshots of Code:



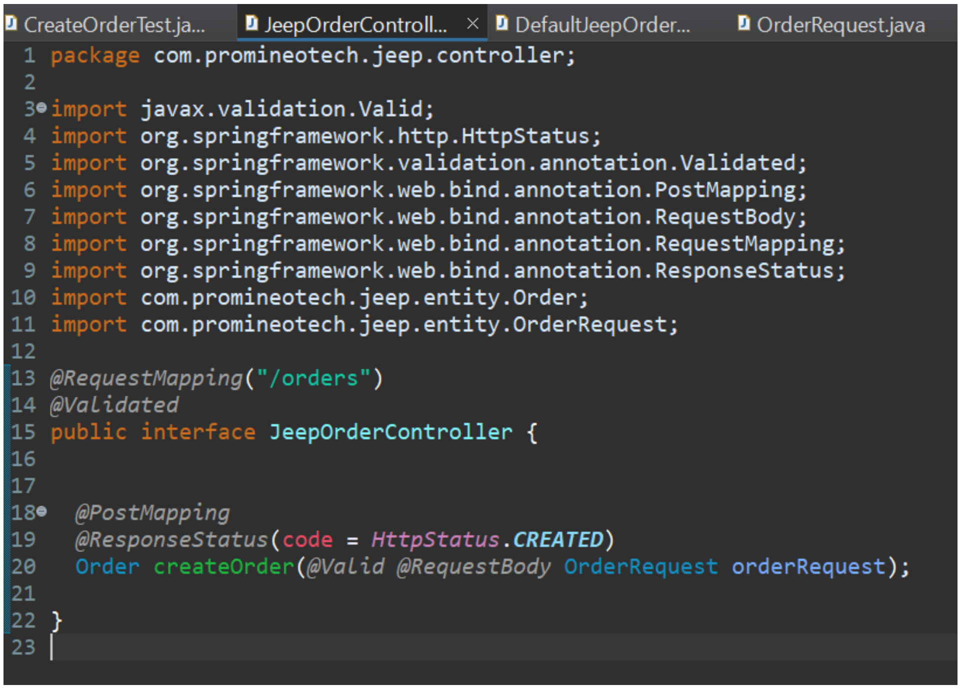
```java
package com.promineotech.jeep.controller;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@Sql(scripts = {
    "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
    "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
    config = @SqlConfig(encoding = "utf-8"))
class CreateOrderTest {

  @LocalServerPort
  private int serverPort;

  @Autowired
  private TestRestTemplate restTemplate;


  @Test
  void testCreateOrderReturnsSuccess201() {
    String body = createOrderBody();
    String uri = String.format("http://localhost:%d/orders", serverPort);

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);


    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

    ResponseEntity<Order> response = restTemplate.exchange(uri,
        HttpMethod.POST, bodyEntity, Order.class);
```



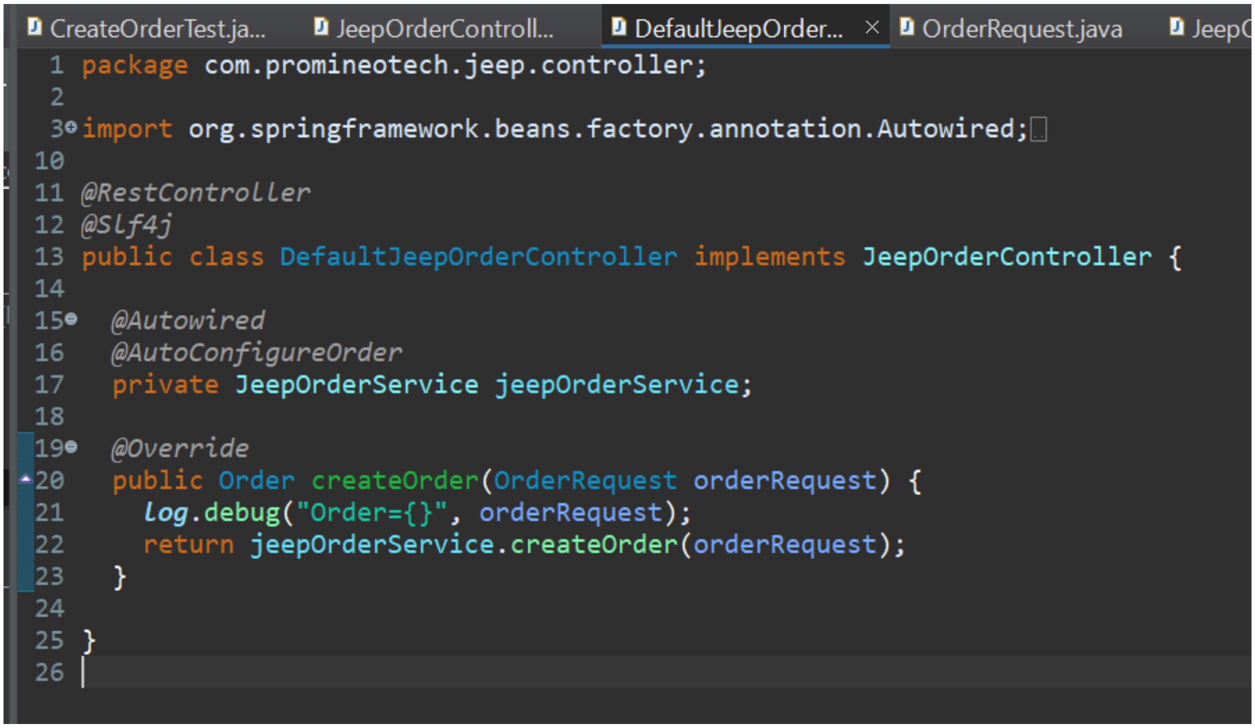
```java
    ResponseEntity<Order> response = restTemplate.exchange(uri,
        HttpMethod.POST, bodyEntity, Order.class);

    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);

  }

  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"
```



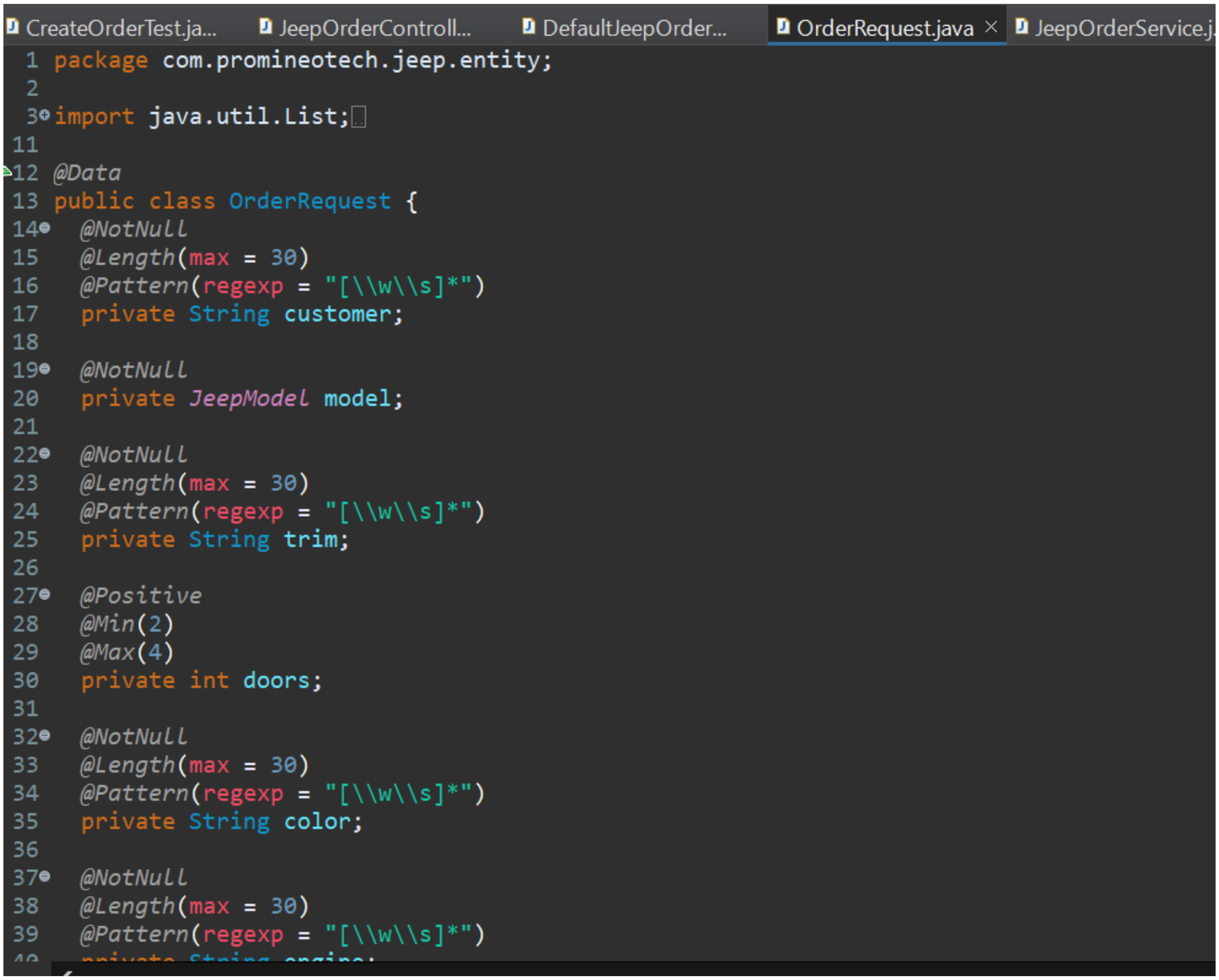
```java
  }

  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
  }






}
```



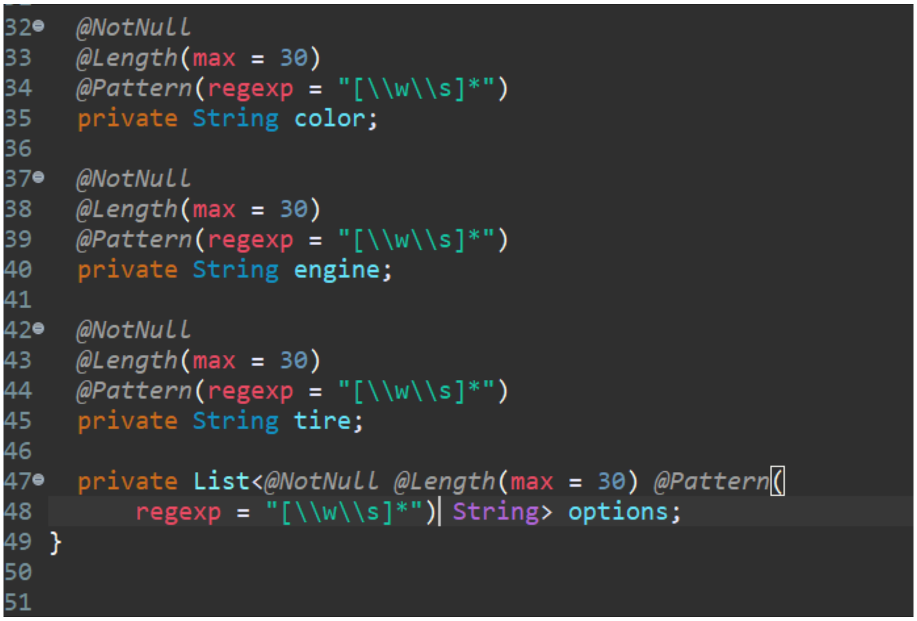
```java
package com.promineotech.jeep.controller;

import javax.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import com.promineotech.jeep.entity.Order;
import com.promineotech.jeep.entity.OrderRequest;

@RequestMapping("/orders")
@Validated
public interface JeepOrderController {


  @PostMapping
  @ResponseStatus(code = HttpStatus.CREATED)
  Order createOrder(@Valid @RequestBody OrderRequest orderRequest);

}

```



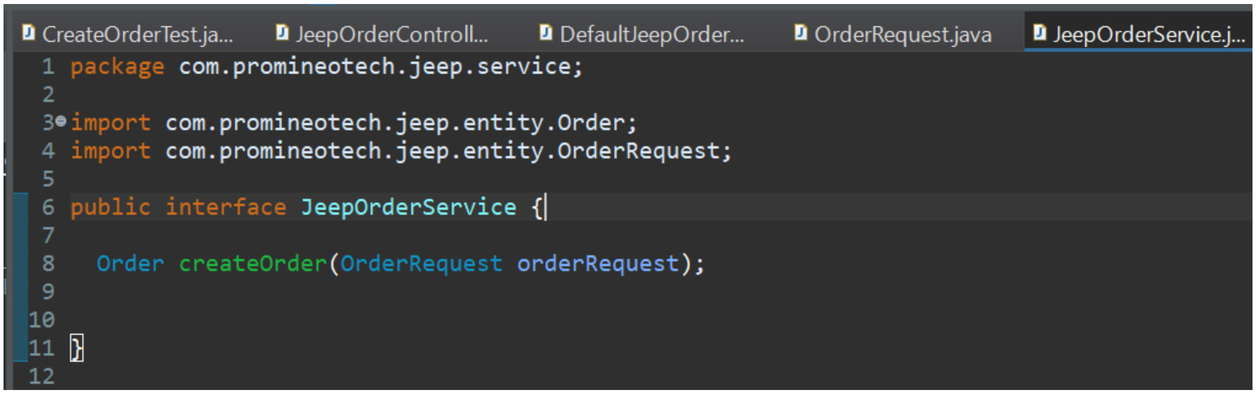
```java
package com.promineotech.jeep.controller;

import org.springframework.beans.factory.annotation.Autowired;

@RestController
@Slf4j
public class DefaultJeepOrderController implements JeepOrderController {

  @Autowired
  @AutoConfigureOrder
  private JeepOrderService jeepOrderService;

  @Override
  public Order createOrder(OrderRequest orderRequest) {
    log.debug("Order={}", orderRequest);
    return jeepOrderService.createOrder(orderRequest);
  }

}
```



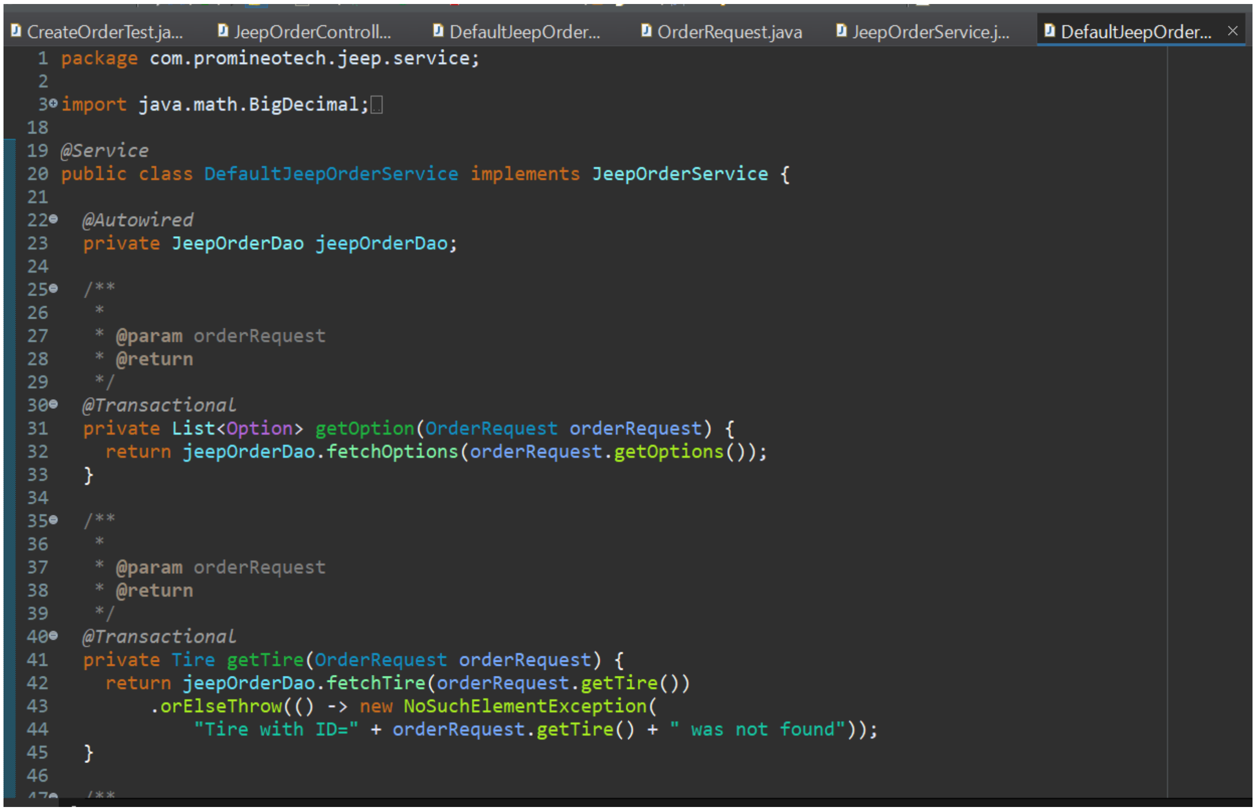
```
package com.promineotech.jeep.entity;

import java.util.List;

@Data
public class OrderRequest {
  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String customer;

  @NotNull
  private JeepModel model;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String trim;

  @Positive
  @Min(2)
  @Max(4)
  private int doors;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String color;

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



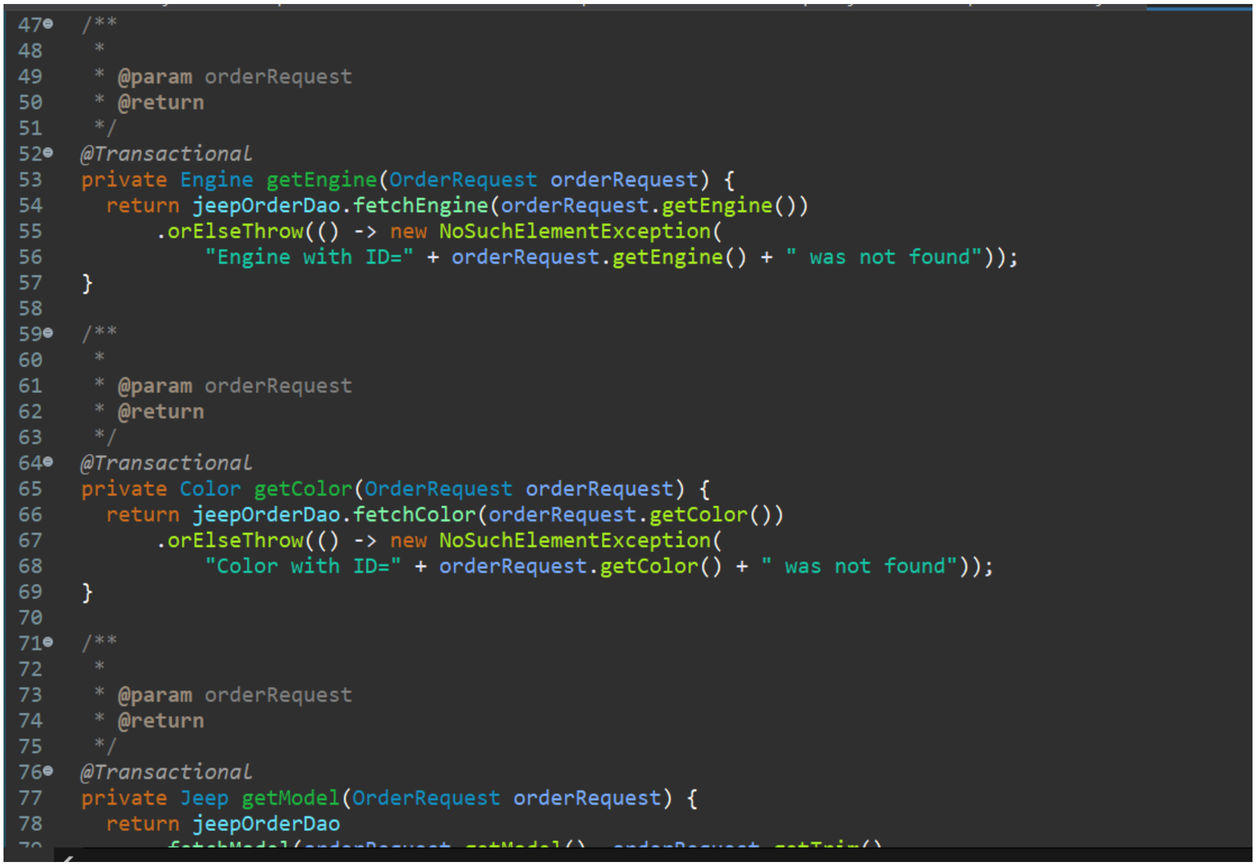
```
  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String color;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String engine;

  @NotNull
  @Length(max = 30)
  @Pattern(regexp = "[\\w\\s]*")
  private String tire;

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



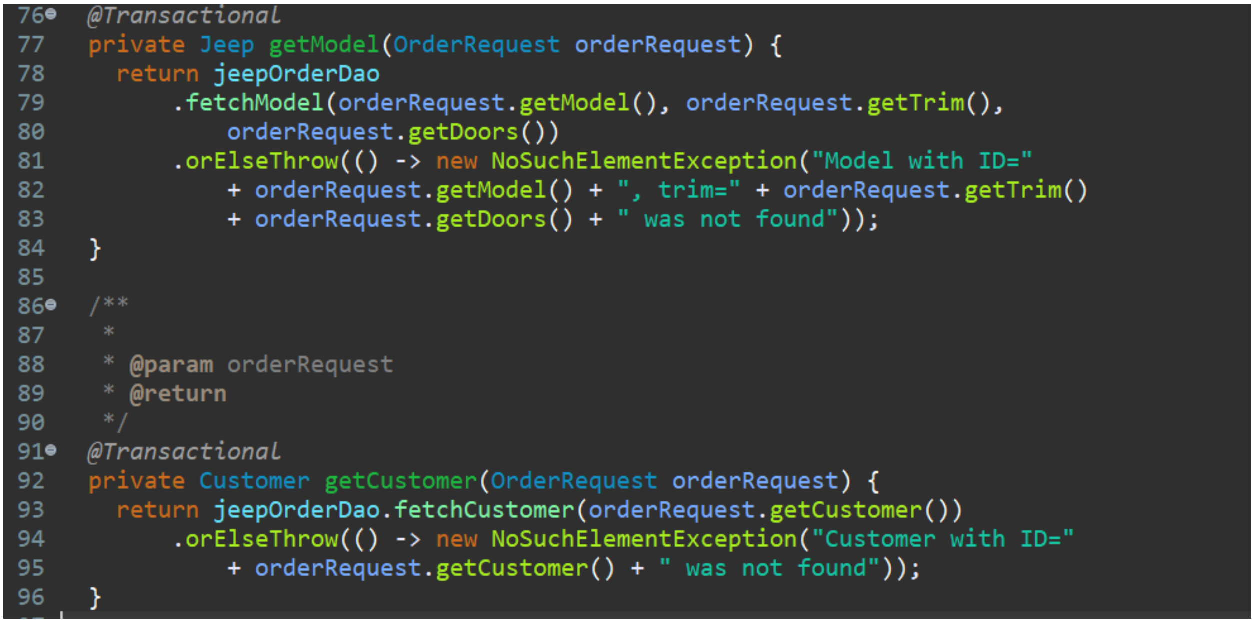
```java
package com.promineotech.jeep.service;

import com.promineotech.jeep.entity.Order;
import com.promineotech.jeep.entity.OrderRequest;

public interface JeepOrderService {

  Order createOrder(OrderRequest orderRequest);


}

```



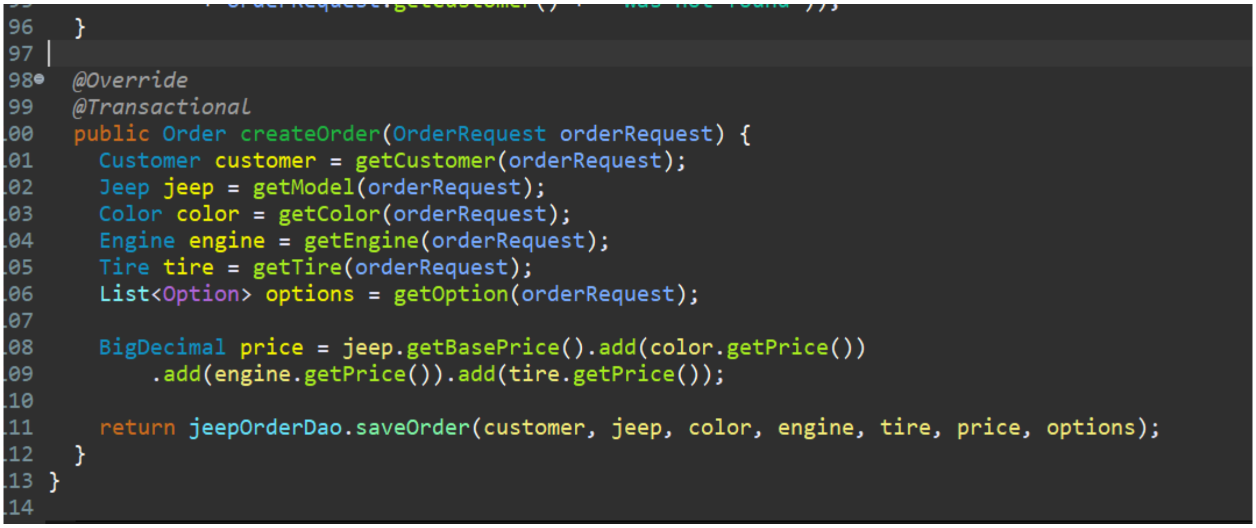
```java
package com.promineotech.jeep.service;

import java.math.BigDecimal;

@Service
public class DefaultJeepOrderService implements JeepOrderService {

  @Autowired
  private JeepOrderDao jeepOrderDao;

  /**
   *
   * @param orderRequest
   * @return
   */
  @Transactional
  private List<Option> getOption(OrderRequest orderRequest) {
    return jeepOrderDao.fetchOptions(orderRequest.getOptions());
  }

  /**
   *
   * @param orderRequest
   * @return
   */
  @Transactional
  private Tire getTire(OrderRequest orderRequest) {
    return jeepOrderDao.fetchTire(orderRequest.getTire())
        .orElseThrow(() -> new NoSuchElementException(
            "Tire with ID=" + orderRequest.getTire() + " was not found"));
  }

  /**
```



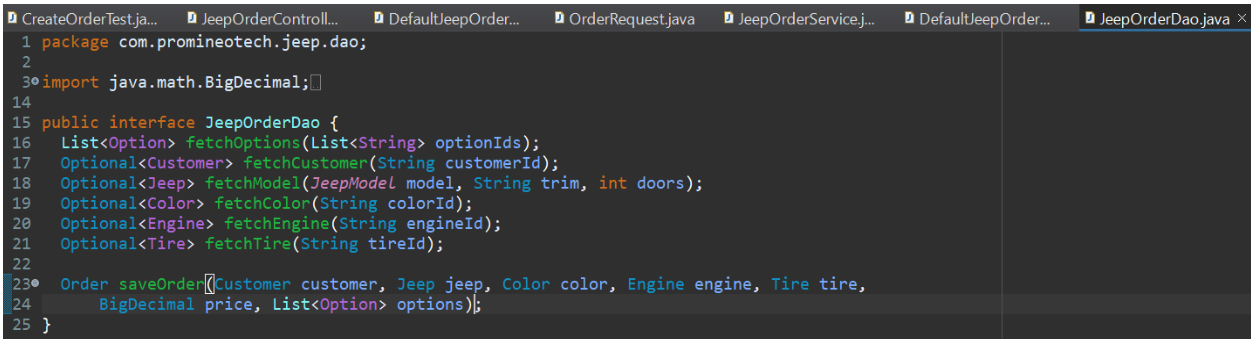
```java
/**
 *
 * @param orderRequest
 * @return
 */
@Transactional
private Engine getEngine(OrderRequest orderRequest) {
  return jeepOrderDao.fetchEngine(orderRequest.getEngine())
      .orElseThrow(() -> new NoSuchElementException(
          "Engine with ID=" + orderRequest.getEngine() + " was not found"));
}

/**
 *
 * @param orderRequest
 * @return
 */
@Transactional
private Color getColor(OrderRequest orderRequest) {
  return jeepOrderDao.fetchColor(orderRequest.getColor())
      .orElseThrow(() -> new NoSuchElementException(
          "Color with ID=" + orderRequest.getColor() + " was not found"));
}

/**
 *
 * @param orderRequest
 * @return
 */
@Transactional
private Jeep getModel(OrderRequest orderRequest) {
  return jeepOrderDao
      .fetchModel(orderRequest.getModel(), orderRequest.getTrim(),
```



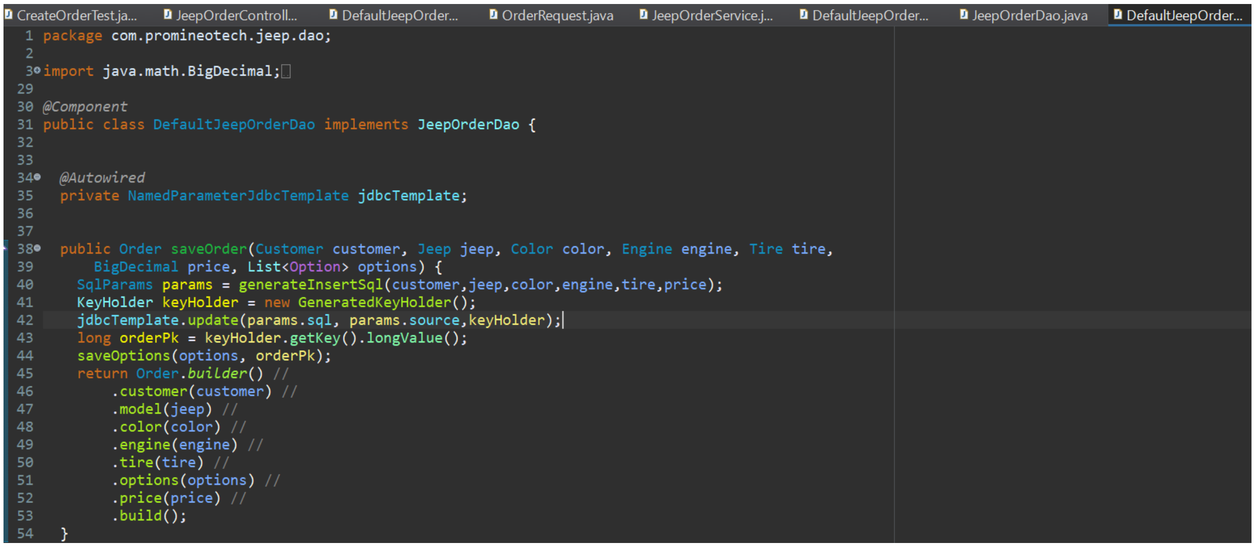
```java
@Transactional
private Jeep getModel(OrderRequest orderRequest) {
  return jeepOrderDao
      .fetchModel(orderRequest.getModel(), orderRequest.getTrim(),
          orderRequest.getDoors())
      .orElseThrow(() -> new NoSuchElementException("Model with ID="
          + orderRequest.getModel() + ", trim=" + orderRequest.getTrim()
          + orderRequest.getDoors() + " was not found"));
}

/**
 *
 * @param orderRequest
 * @return
 */
@Transactional
private Customer getCustomer(OrderRequest orderRequest) {
  return jeepOrderDao.fetchCustomer(orderRequest.getCustomer())
      .orElseThrow(() -> new NoSuchElementException("Customer with ID="
          + orderRequest.getCustomer() + " was not found"));
}
```



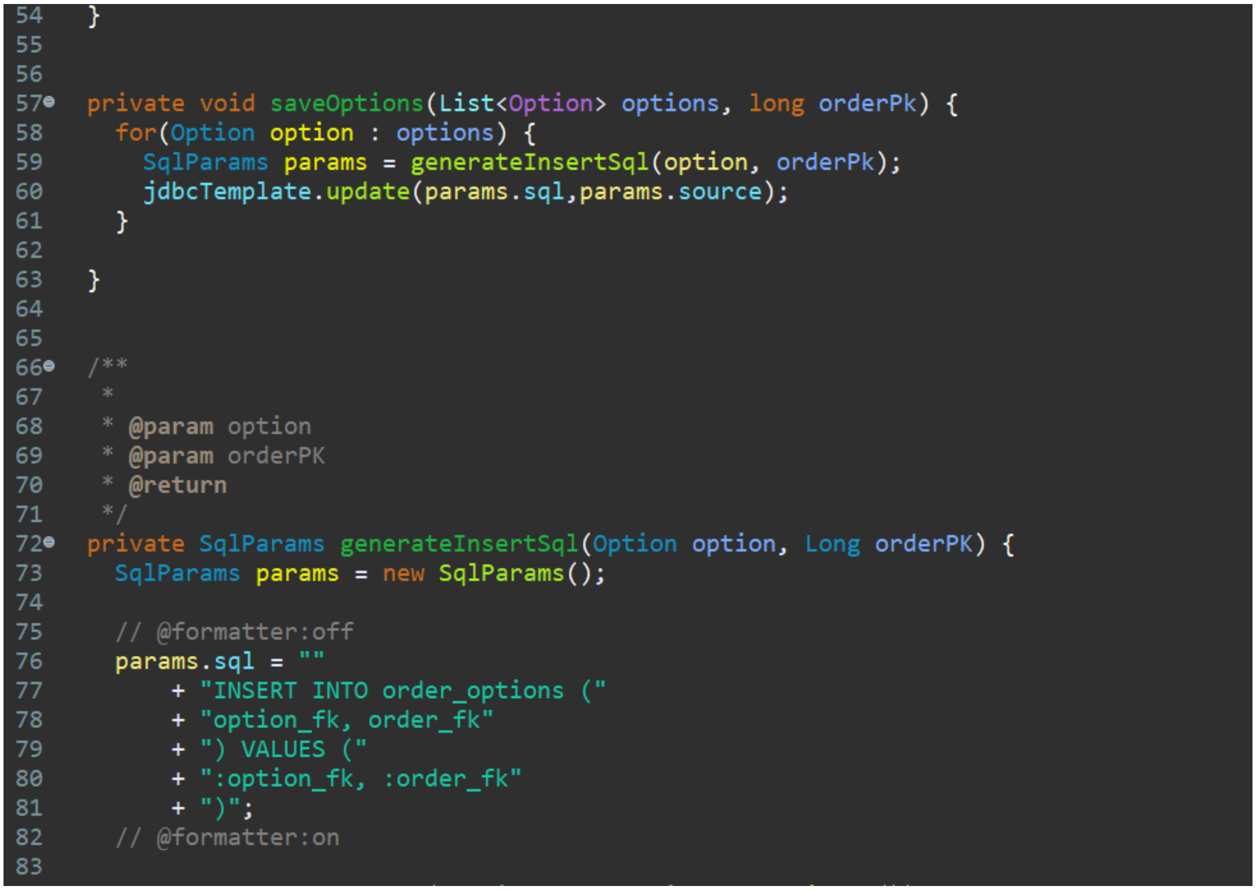
```java
          + orderRequest.getCustomer() + " was not found"));
  }

  @Override
  @Transactional
  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());

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



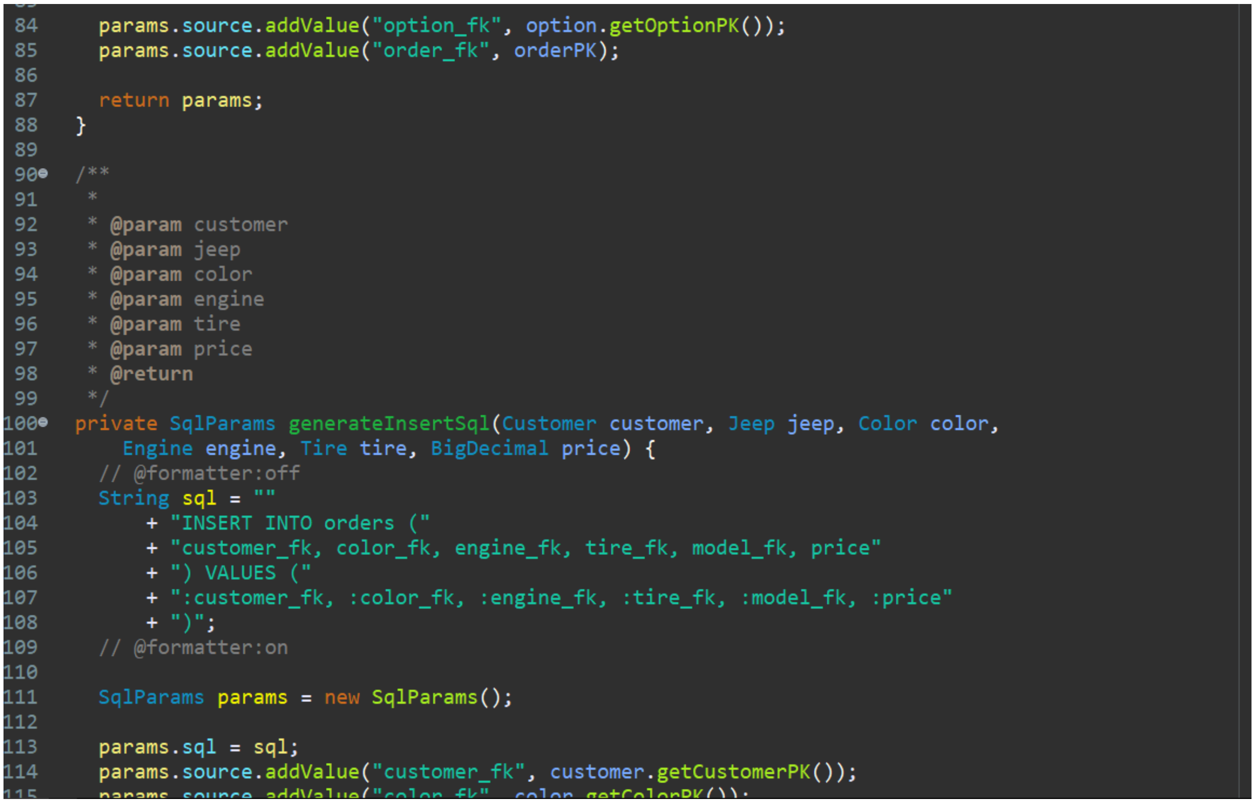
```java
package com.promineotech.jeep.dao;

import java.math.BigDecimal;

public interface JeepOrderDao {
  List<Option> fetchOptions(List<String> optionIds);
  Optional<Customer> fetchCustomer(String customerId);
  Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
  Optional<Color> fetchColor(String colorId);
  Optional<Engine> fetchEngine(String engineId);
  Optional<Tire> fetchTire(String tireId);

  Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
      BigDecimal price, List<Option> options);
}
```



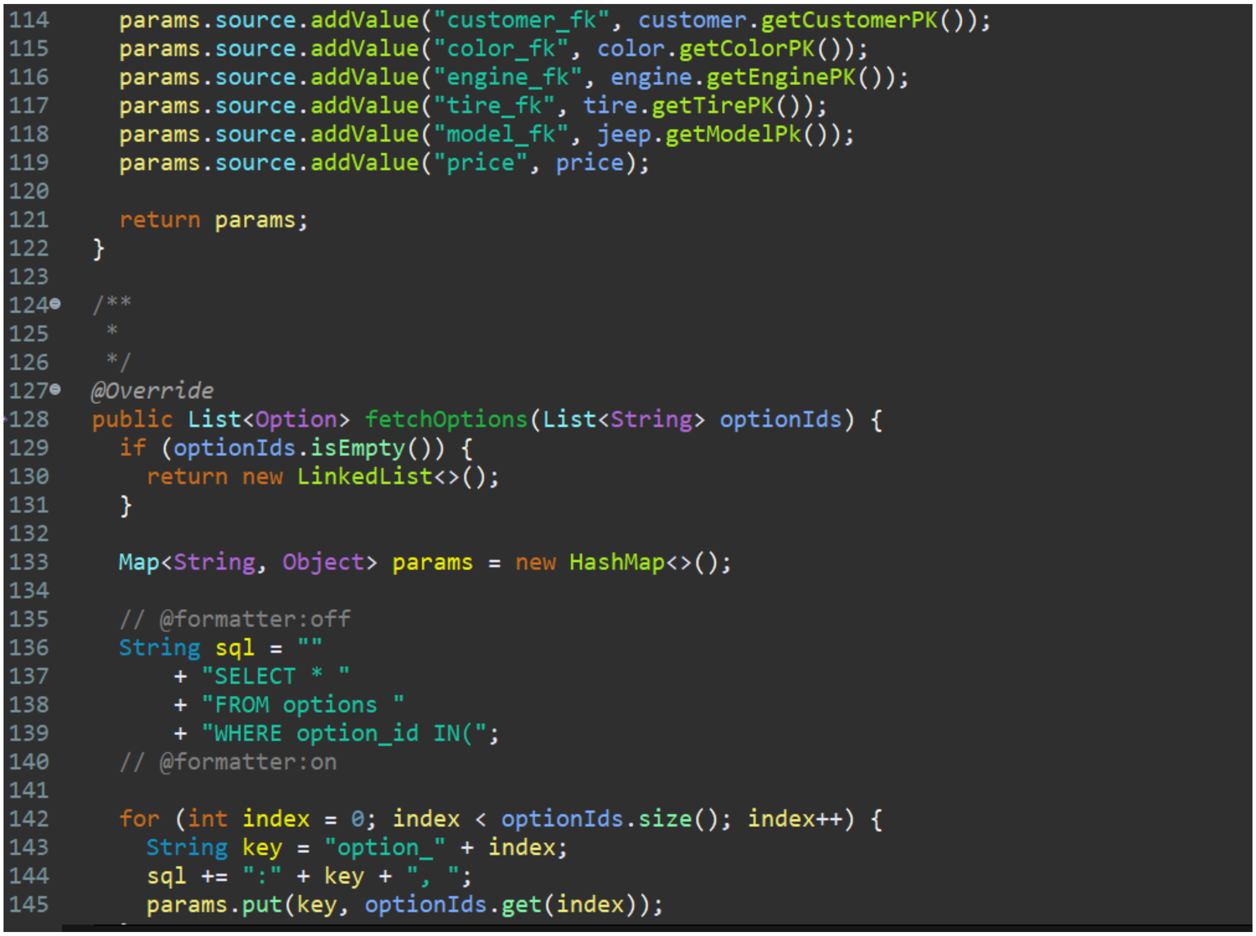
```java
package com.promineotech.jeep.dao;

import java.math.BigDecimal;

@Component
public class DefaultJeepOrderDao implements JeepOrderDao {


  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;


  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);
    return Order.builder() //
        .customer(customer) //
        .model(jeep) //
        .color(color) //
        .engine(engine) //
        .tire(tire) //
        .options(options) //
        .price(price) //
        .build();
  }
```



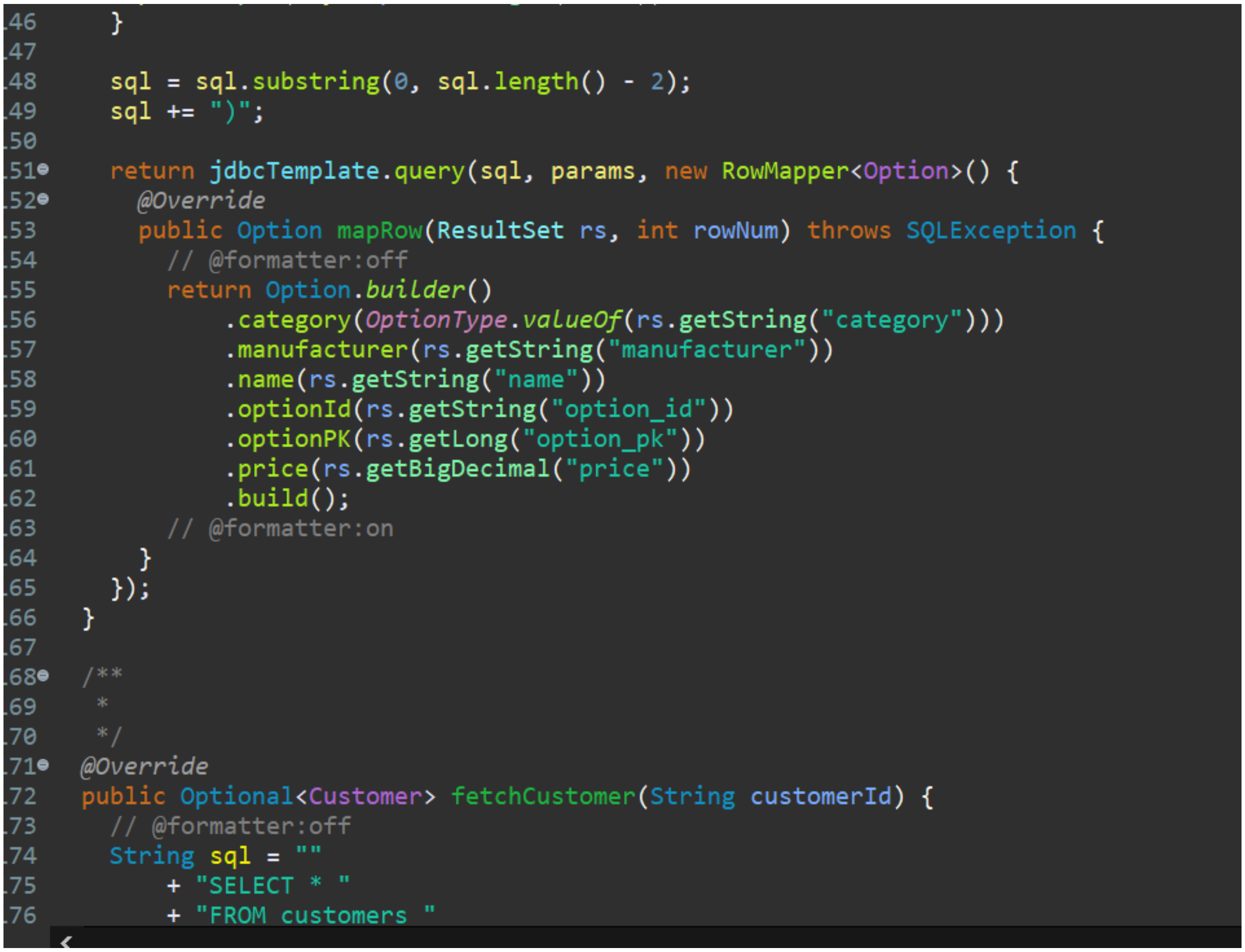
```java
  }


  private void saveOptions(List<Option> options, long orderPk) {
    for(Option option : options) {
      SqlParams params = generateInsertSql(option, orderPk);
      jdbcTemplate.update(params.sql,params.source);
    }

  }


  /**
   *
   * @param option
   * @param orderPK
   * @return
   */
  private SqlParams generateInsertSql(Option option, Long orderPK) {
    SqlParams params = new SqlParams();

    // @formatter:off
    params.sql = ""
        + "INSERT INTO order_options ("
        + "option_fk, order_fk"
        + ") VALUES ("
        + ":option_fk, :order_fk"
        + ")";
    // @formatter:on

```



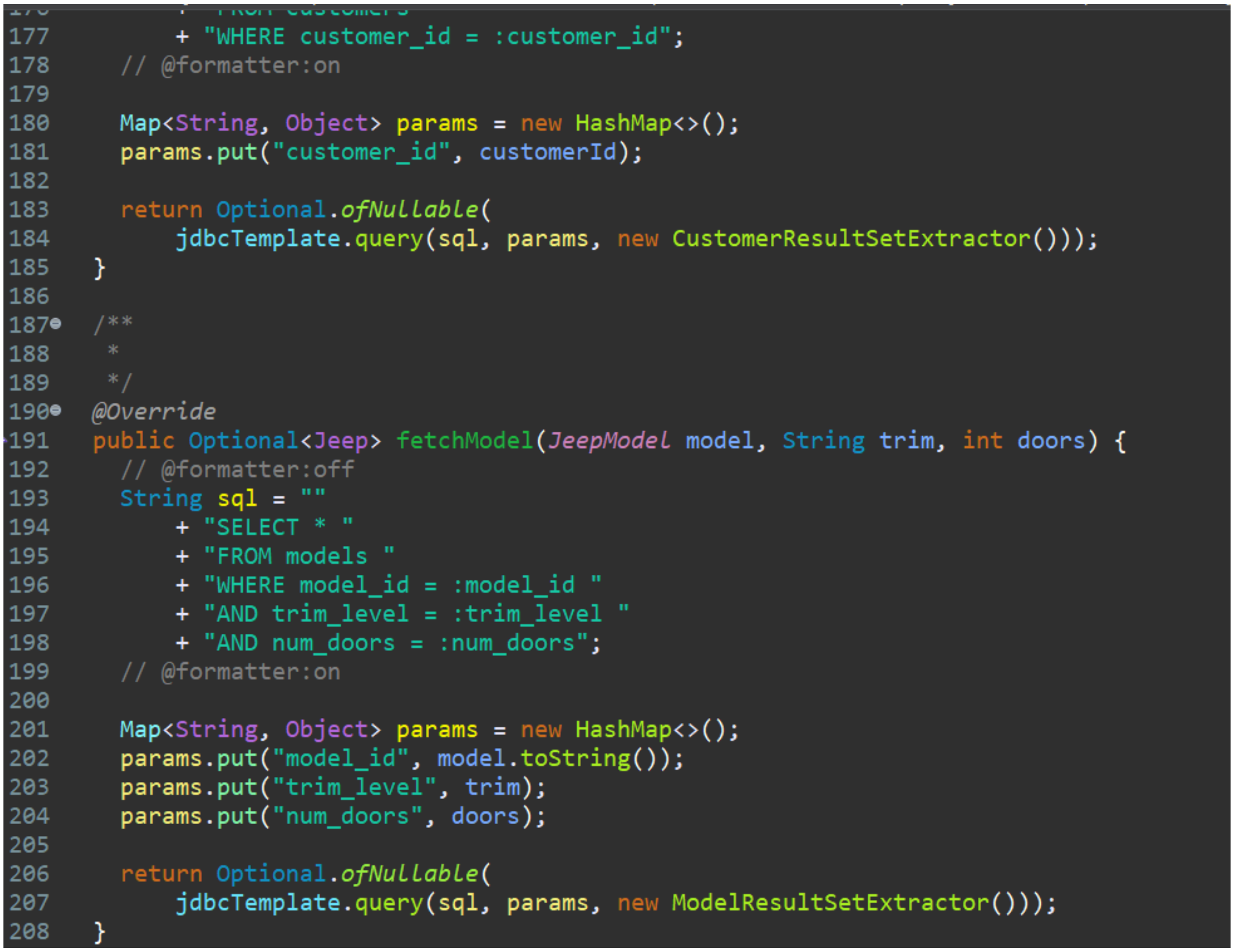
```java
    params.source.addValue("option_fk", option.getOptionPK());
    params.source.addValue("order_fk", orderPK);

    return params;
  }

  /**
   *
   * @param customer
   * @param jeep
   * @param color
   * @param engine
   * @param tire
   * @param price
   * @return
   */
  private SqlParams generateInsertSql(Customer customer, Jeep jeep, Color color,
      Engine engine, Tire tire, BigDecimal price) {
    // @formatter:off
    String sql = ""
        + "INSERT INTO orders ("
        + "customer_fk, color_fk, engine_fk, tire_fk, model_fk, price"
        + ") VALUES ("
        + ":customer_fk, :color_fk, :engine_fk, :tire_fk, :model_fk, :price"
        + ")";
    // @formatter:on

    SqlParams params = new SqlParams();

    params.sql = sql;
    params.source.addValue("customer_fk", customer.getCustomerPK());
    params.source.addValue("color_fk", color.getColorPK());
```



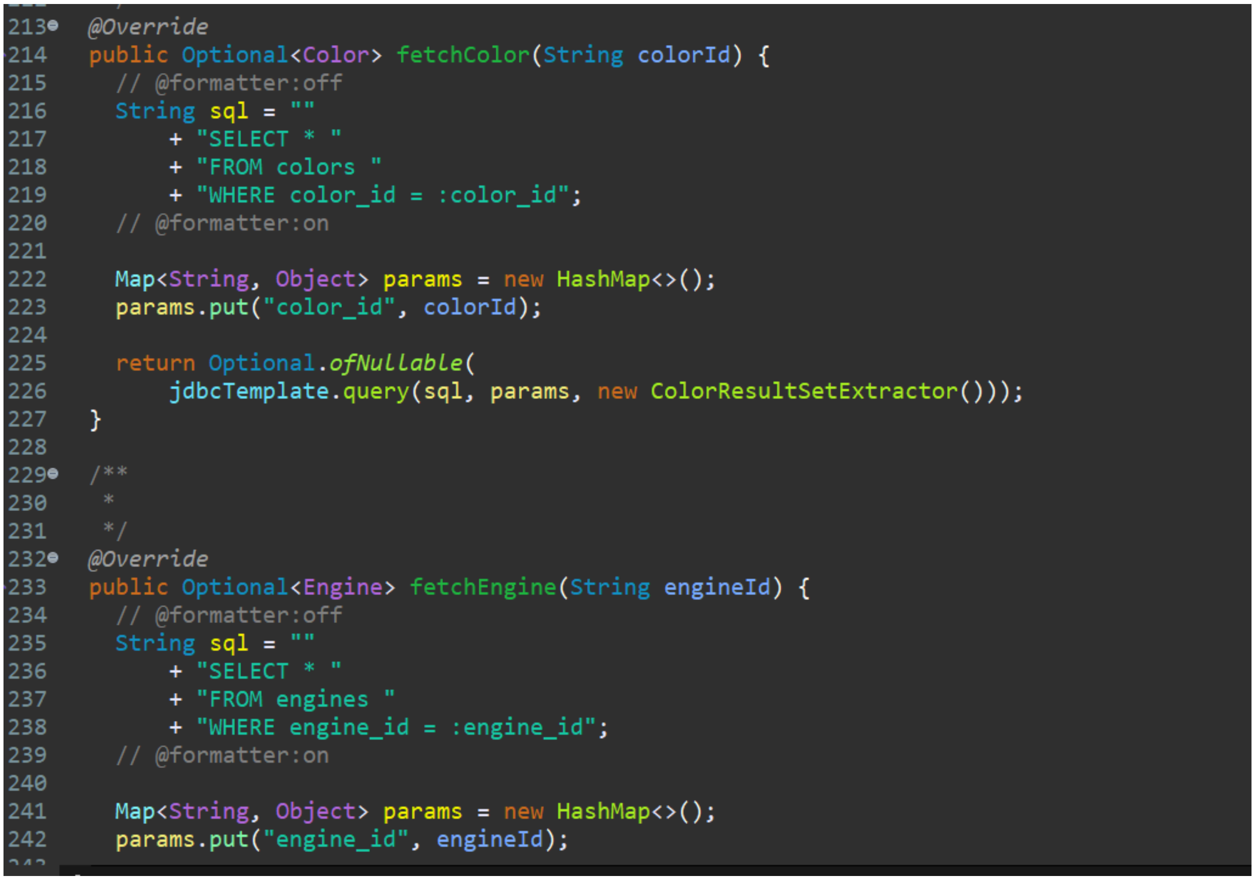
```java
    params.source.addValue("customer_fk", customer.getCustomerPK());
    params.source.addValue("color_fk", color.getColorPK());
    params.source.addValue("engine_fk", engine.getEnginePK());
    params.source.addValue("tire_fk", tire.getTirePK());
    params.source.addValue("model_fk", jeep.getModelPk());
    params.source.addValue("price", price);

    return params;
  }

  /**
   *
   */
  @Override
  public List<Option> fetchOptions(List<String> optionIds) {
    if (optionIds.isEmpty()) {
      return new LinkedList<>();
    }

    Map<String, Object> params = new HashMap<>();

    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM options "
        + "WHERE option_id IN(";
    // @formatter:on

    for (int index = 0; index < optionIds.size(); index++) {
      String key = "option_" + index;
      sql += ":" + key + ", ";
      params.put(key, optionIds.get(index));
```



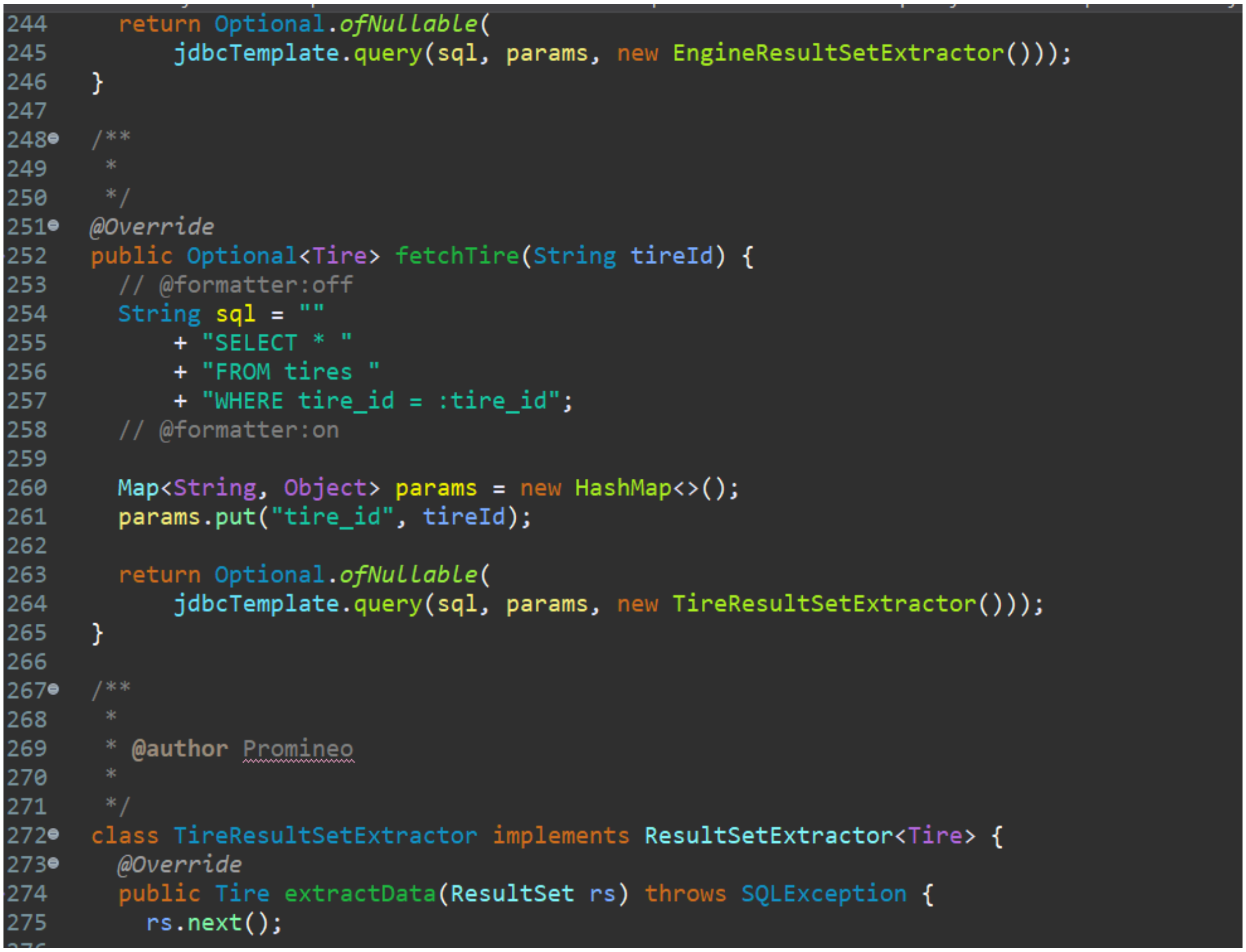
```
  }

  sql = sql.substring(0, sql.length() - 2);
  sql += ")";

  return jdbcTemplate.query(sql, params, new RowMapper<Option>() {
    @Override
    public Option mapRow(ResultSet rs, int rowNum) throws SQLException {
      // @formatter:off
      return Option.builder()
          .category(OptionType.valueOf(rs.getString("category")))
          .manufacturer(rs.getString("manufacturer"))
          .name(rs.getString("name"))
          .optionId(rs.getString("option_id"))
          .optionPK(rs.getLong("option_pk"))
          .price(rs.getBigDecimal("price"))
          .build();
      // @formatter:on
    }
  });
}

/**
 *
 */
@Override
public Optional<Customer> fetchCustomer(String customerId) {
  // @formatter:off
  String sql = ""
      + "SELECT * "
      + "FROM customers "
```



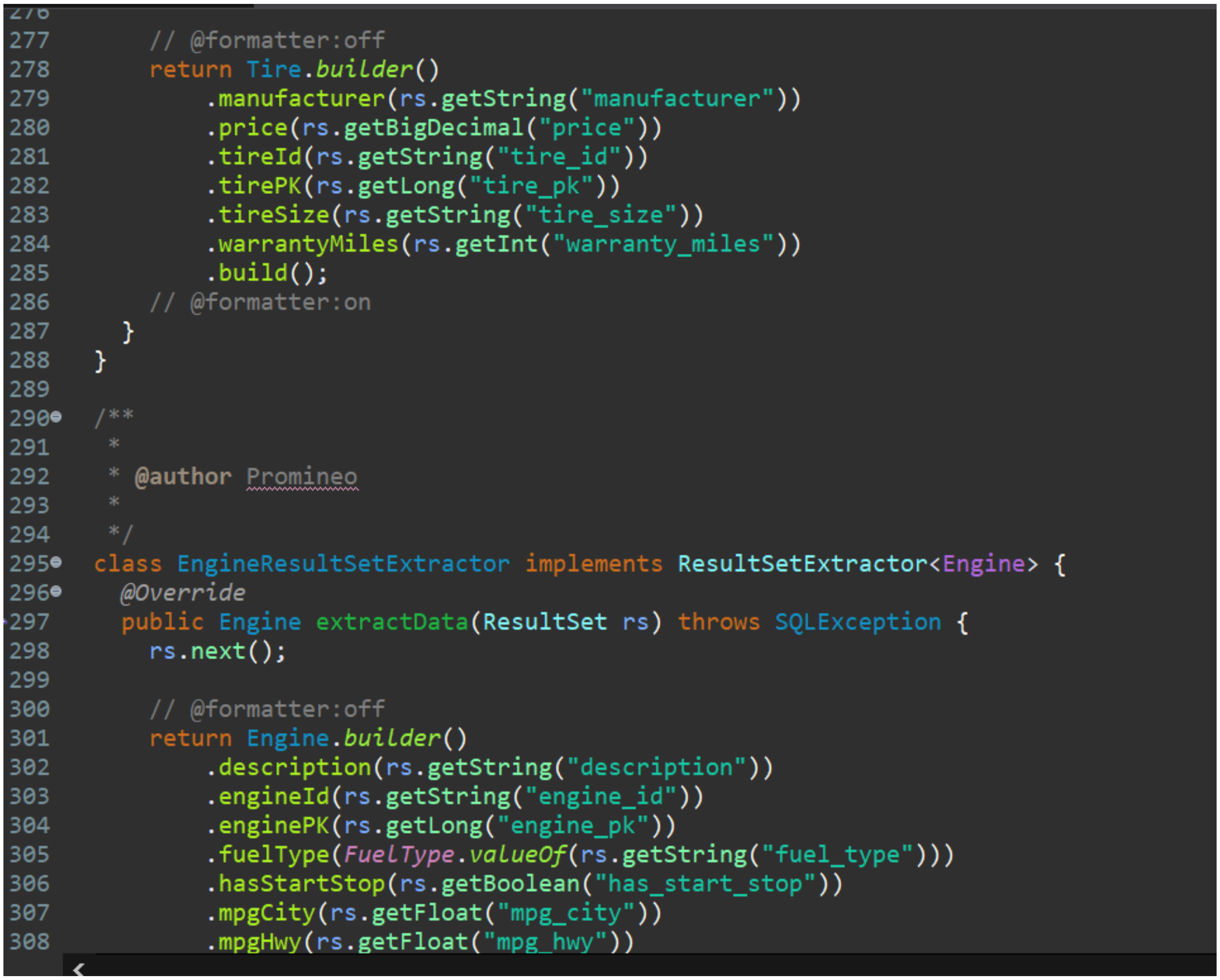
```
        + "FROM customers "
        + "WHERE customer_id = :customer_id";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("customer_id", customerId);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new CustomerResultSetExtractor()));
  }

  /**
   *
   */
  @Override
  public Optional<Jeep> fetchModel(JeepModel model, String trim, int doors) {
    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM models "
        + "WHERE model_id = :model_id "
        + "AND trim_level = :trim_level "
        + "AND num_doors = :num_doors";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("model_id", model.toString());
    params.put("trim_level", trim);
    params.put("num_doors", doors);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new ModelResultSetExtractor()));
  }
```



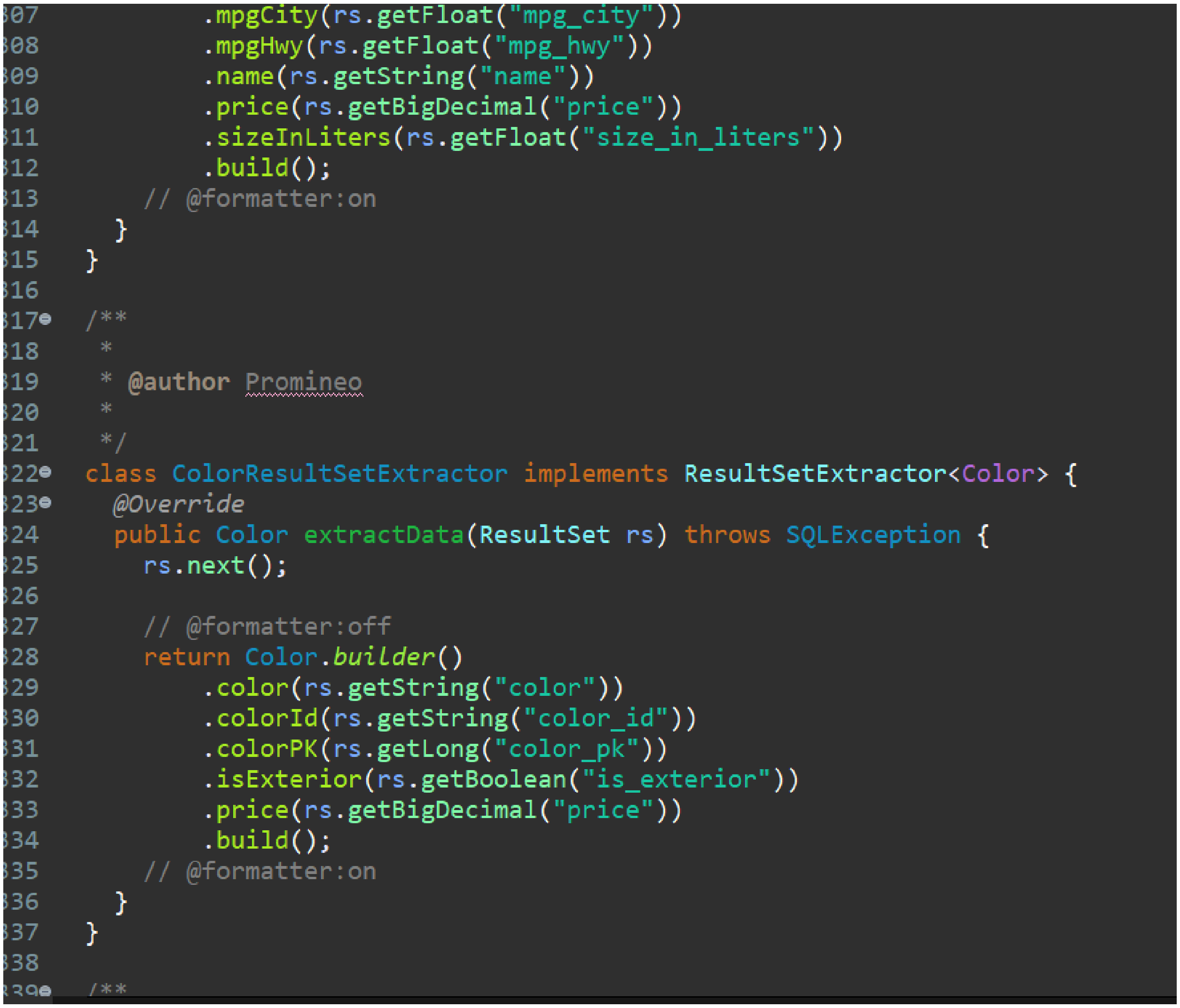
```java
  @Override
  public Optional<Color> fetchColor(String colorId) {
    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM colors "
        + "WHERE color_id = :color_id";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("color_id", colorId);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new ColorResultSetExtractor()));
  }

  /**
   *
   */
  @Override
  public Optional<Engine> fetchEngine(String engineId) {
    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM engines "
        + "WHERE engine_id = :engine_id";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("engine_id", engineId);
```



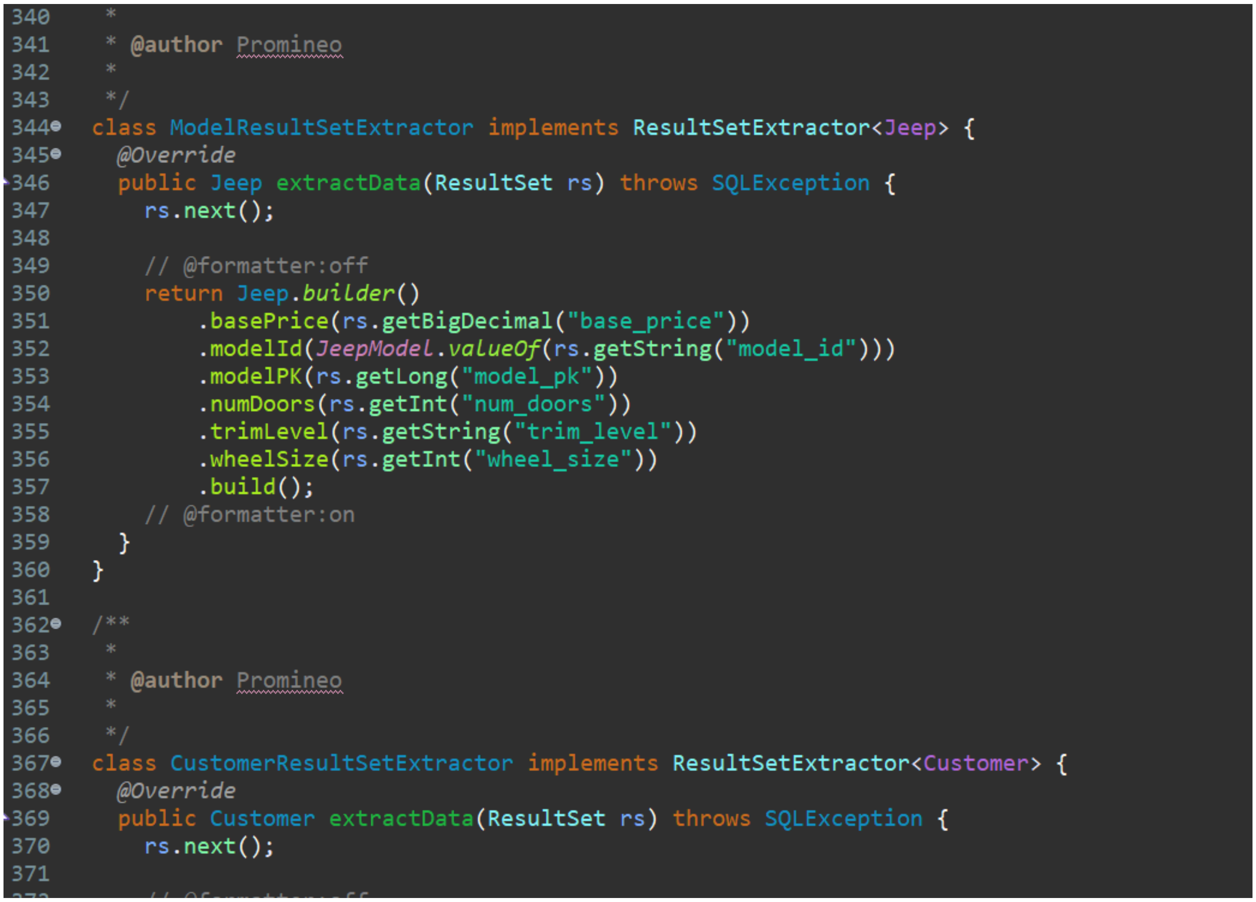
```
    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new EngineResultSetExtractor()));
  }

  /**
   *
   */
  @Override
  public Optional<Tire> fetchTire(String tireId) {
    // @formatter:off
    String sql = ""
        + "SELECT * "
        + "FROM tires "
        + "WHERE tire_id = :tire_id";
    // @formatter:on

    Map<String, Object> params = new HashMap<>();
    params.put("tire_id", tireId);

    return Optional.ofNullable(
        jdbcTemplate.query(sql, params, new TireResultSetExtractor()));
  }

  /**
   *
   * @author Promineo
   *
   */
  class TireResultSetExtractor implements ResultSetExtractor<Tire> {
    @Override
    public Tire extractData(ResultSet rs) throws SQLException {
      rs.next();
```



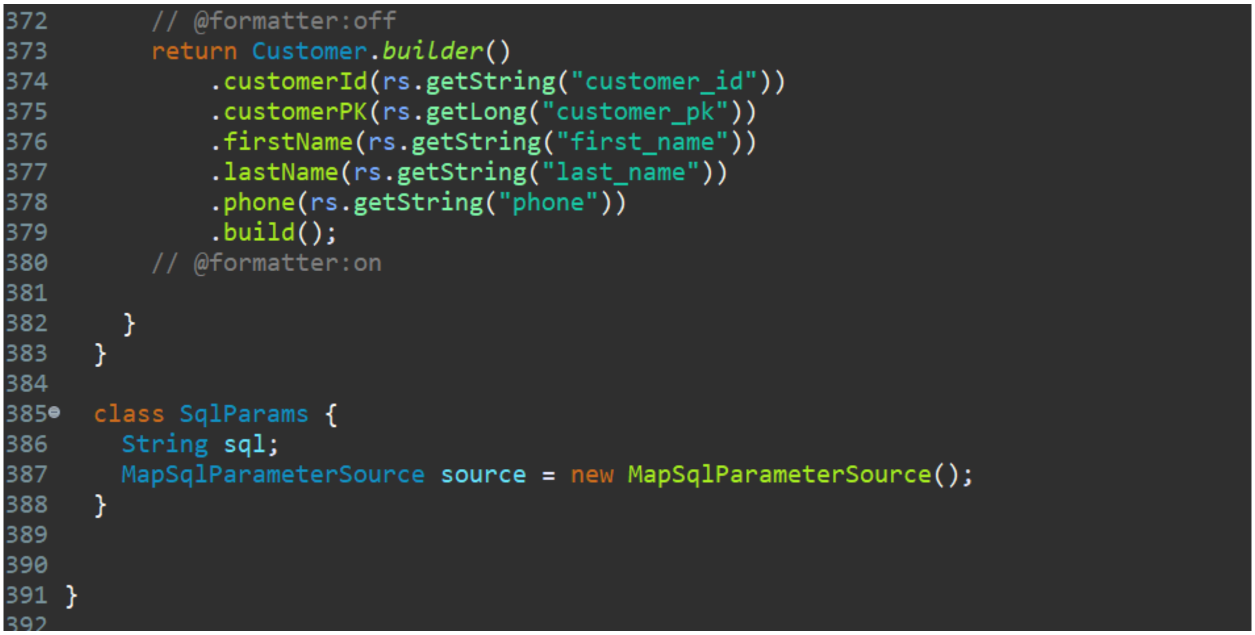
```
    // @formatter:off
    return Tire.builder()
        .manufacturer(rs.getString("manufacturer"))
        .price(rs.getBigDecimal("price"))
        .tireId(rs.getString("tire_id"))
        .tirePK(rs.getLong("tire_pk"))
        .tireSize(rs.getString("tire_size"))
        .warrantyMiles(rs.getInt("warranty_miles"))
        .build();
    // @formatter:on
  }
}

/**
 *
 * @author Promineo
 *
 */
class EngineResultSetExtractor implements ResultSetExtractor<Engine> {
  @Override
  public Engine extractData(ResultSet rs) throws SQLException {
    rs.next();

    // @formatter:off
    return Engine.builder()
        .description(rs.getString("description"))
        .engineId(rs.getString("engine_id"))
        .enginePK(rs.getLong("engine_pk"))
        .fuelType(FuelType.valueOf(rs.getString("fuel_type")))
        .hasStartStop(rs.getBoolean("has_start_stop"))
        .mpgCity(rs.getFloat("mpg_city"))
        .mpgHwy(rs.getFloat("mpg_hwy"))
```



```
        .mpgCity(rs.getFloat("mpg_city"))
        .mpgHwy(rs.getFloat("mpg_hwy"))
        .name(rs.getString("name"))
        .price(rs.getBigDecimal("price"))
        .sizeInLiters(rs.getFloat("size_in_liters"))
        .build();
    // @formatter:on
  }
}

/**
 *
 * @author Promineo
 *
 */
class ColorResultSetExtractor implements ResultSetExtractor<Color> {
  @Override
  public Color extractData(ResultSet rs) throws SQLException {
    rs.next();

    // @formatter:off
    return Color.builder()
        .color(rs.getString("color"))
        .colorId(rs.getString("color_id"))
        .colorPK(rs.getLong("color_pk"))
        .isExterior(rs.getBoolean("is_exterior"))
        .price(rs.getBigDecimal("price"))
        .build();
    // @formatter:on
  }
}

/**
```

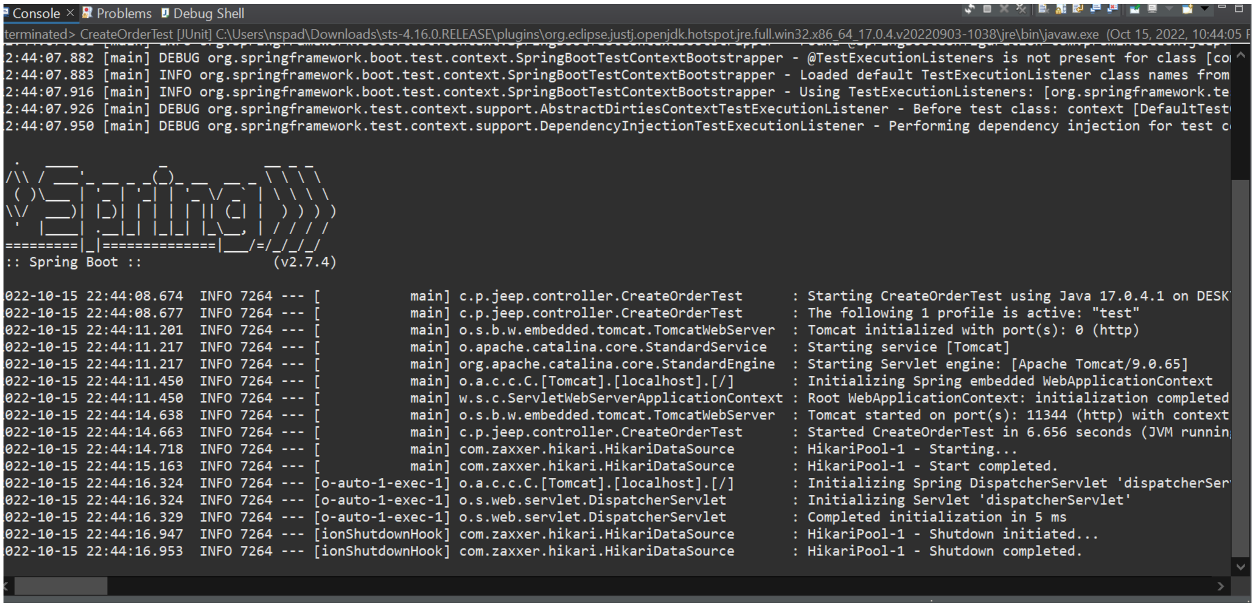


```java
   *
   * @author Promineo
   *
   */
  class ModelResultSetExtractor implements ResultSetExtractor<Jeep> {
    @Override
    public Jeep extractData(ResultSet rs) throws SQLException {
      rs.next();

      // @formatter:off
      return Jeep.builder()
          .basePrice(rs.getBigDecimal("base_price"))
          .modelId(JeepModel.valueOf(rs.getString("model_id")))
          .modelPK(rs.getLong("model_pk"))
          .numDoors(rs.getInt("num_doors"))
          .trimLevel(rs.getString("trim_level"))
          .wheelSize(rs.getInt("wheel_size"))
          .build();
      // @formatter:on
    }
  }

  /**
   *
   * @author Promineo
   *
   */
  class CustomerResultSetExtractor implements ResultSetExtractor<Customer> {
    @Override
    public Customer extractData(ResultSet rs) throws SQLException {
      rs.next();

```



```java
      // @formatter:off
      return Customer.builder()
          .customerId(rs.getString("customer_id"))
          .customerPK(rs.getLong("customer_pk"))
          .firstName(rs.getString("first_name"))
          .lastName(rs.getString("last_name"))
          .phone(rs.getString("phone"))
          .build();
      // @formatter:on

    }
  }

  class SqlParams {
    String sql;
    MapSqlParameterSource source = new MapSqlParameterSource();
  }


}
```

Screenshots of Running Application:



```
terminated> CreateOrderTest [JUnit] C:\Users\nspad\Downloads\sts-4.16.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe  (Oct 15, 2022, 10:44:05 F
2:44:07.882 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper - @TestExecutionListeners is not present for class [co
2:44:07.883 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Loaded default TestExecutionListener class names from
2:44:07.916 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.te
2:44:07.926 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTest
2:44:07.950 [main] DEBUG org.springframework.test.context.support.DependencyInjectionTestExecutionListener - Performing dependency injection for test c


 .   ____          _            __ _ _
/\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
 ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
\\/  ___)| |_)| | | | | || (_| |  ) ) ) )
 '  |____| .__|_| |_|_| |_\__, | / / / /
=========|_|==============|___/=/_/_/_/
:: Spring Boot ::                (v2.7.4)

022-10-15 22:44:08.674  INFO 7264 --- [           main] c.p.jeep.controller.CreateOrderTest      : Starting CreateOrderTest using Java 17.0.4.1 on DESK
022-10-15 22:44:08.677  INFO 7264 --- [           main] c.p.jeep.controller.CreateOrderTest      : The following 1 profile is active: "test"
022-10-15 22:44:11.201  INFO 7264 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 0 (http)
022-10-15 22:44:11.217  INFO 7264 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
022-10-15 22:44:11.217  INFO 7264 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
022-10-15 22:44:11.450  INFO 7264 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
022-10-15 22:44:11.450  INFO 7264 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
022-10-15 22:44:14.638  INFO 7264 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 11344 (http) with context
022-10-15 22:44:14.663  INFO 7264 --- [           main] c.p.jeep.controller.CreateOrderTest      : Started CreateOrderTest in 6.656 seconds (JVM runnin
022-10-15 22:44:14.718  INFO 7264 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
022-10-15 22:44:15.163  INFO 7264 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
022-10-15 22:44:16.324  INFO 7264 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherSer
022-10-15 22:44:16.324  INFO 7264 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
022-10-15 22:44:16.329  INFO 7264 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 5 ms
022-10-15 22:44:16.947  INFO 7264 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
022-10-15 22:44:16.953  INFO 7264 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

URL to GitHub Repository:

https://github.com/tyoung3614/jeep-sales