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. 🖥️

```
29    protected String createOrderBody() {
30        // @formatter:off
31        return "{\n"
32                + "   \"customer\":\"MORISON_LINA\",\n"
33                + "   \"model\":\"WRANGLER\",\n"
34                + "   \"trim\":\"Sport Altitude\",\n"
35                + "   \"doors\":4,\n"
36                + "   \"color\":\"EXT_NACHO\",\n"
37                + "   \"engine\":\"2_0_TURBO\",\n"
38                + "   \"tire\":\"35_TOYO\",\n"
39                + "   \"options\":[\n"
40                + "     \"DOOR_QUAD_4\",\n"
41                + "     \"EXT_AEV_LIFT\",\n"
42                + "     \"EXT_WARN_BUMPER_FRONT\",\n"
43                + "     \"EXT_WARN_BUMPER_REAR\",\n"
44                + "     \"EXT_ARB_COMPRESSOR\",\n"
45                + " ]\n"
46                + "}";
47        // @formatter:on
48    }
```

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() {
    //Given: an order as JSON
    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);

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

    //Then: a 201 status is returned
    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);

    //And: the returned order is correct
}
```

3) In the `controller` sub-package in `src/main/java`, create an interface named `JeepOrderController`. Add `@RequestMapping("/orders")` as a class-level annotation.

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.

CreateOrderTest.java | JeepOrderController.java

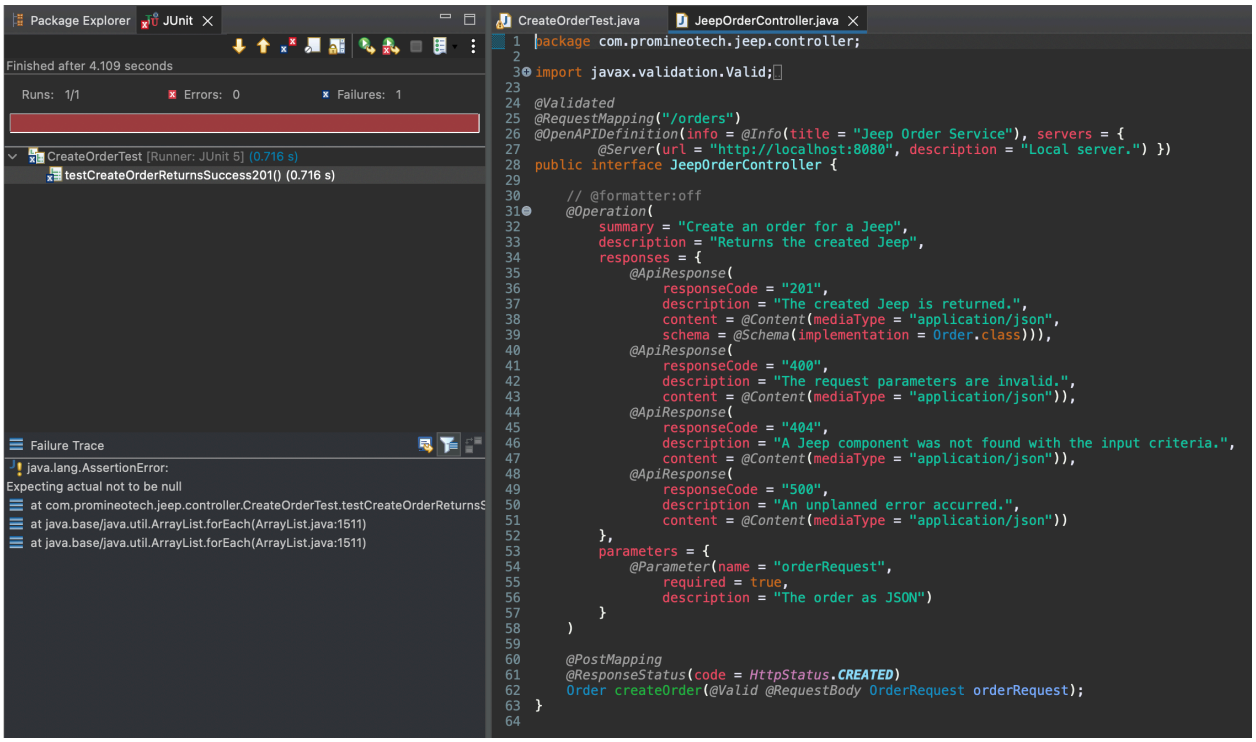
```java
package com.promineotech.jeep.controller;

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;

import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.servers.Server;

@RequestMapping("/orders")
@OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
        @Server(url = "http://localhost:8080", description = "Local server.")})
public interface JeepOrderController {

    // @formatter:off
    @Operation(
        summary = "Create an order for a Jeep",
        description = "Returns the created Jeep",
        responses = {
            @ApiResponse(
                responseCode = "201",
                description = "The created Jeep is returned.",
                content = @Content(mediaType = "application/json",
                schema = @Schema(implementation = Order.class)))
        },
        parameters = {
            @Parameter(name = "orderRequest",
                required = true,
                description = "The order as JSON")
        }
    )

    @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.



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.

CreateOrderTest.java | OrderRequest.java

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

import java.util.List;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Positive;

import org.hibernate.validator.constraints.Length;

import lombok.Data;

@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 tire;

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

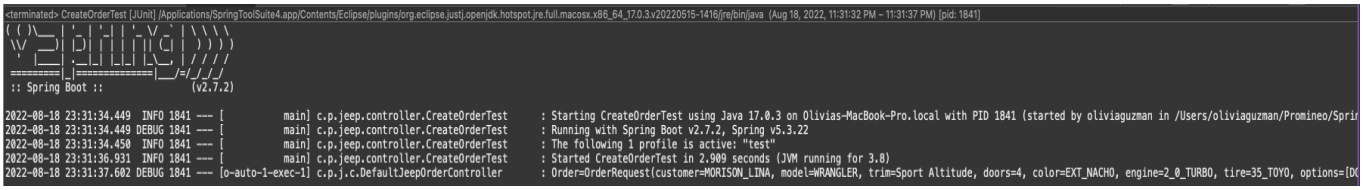
8) In the `jeep.service` sub-package, create the empty (no methods yet) order service interface (named `JeepOrderService`) and implementation (named `DefaultJeepOrderService`).

   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:

```
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.

```
27    @Transactional
28    @Override
29    public Order createOrder(OrderRequest orderRequest) {
30        Customer customer = getCustomer(orderRequest);
31        Jeep jeep = getModel(orderRequest);
32        Color color = getColor(orderRequest);
33        Engine engine = getEngine(orderRequest);
34        Tire tire = getTire(orderRequest);
35        List<Option> options = getOption(orderRequest);
36
37        BigDecimal price = jeep.getBasePrice()
38                .add(color.getPrice())
39                .add(engine.getPrice())
40                .add(tire.getPrice());
41
42        for(Option option : options) {
43            price.add(option.getPrice());
44        }
45
46        return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
47    }
```

b) Write the implementation of the `saveOrder` method in the DAO.

i) Call the supplied `generateInsertSql` method, passing in the customer, jeep, color, engine, tire and price. Assign the return value of the method to a `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)
```

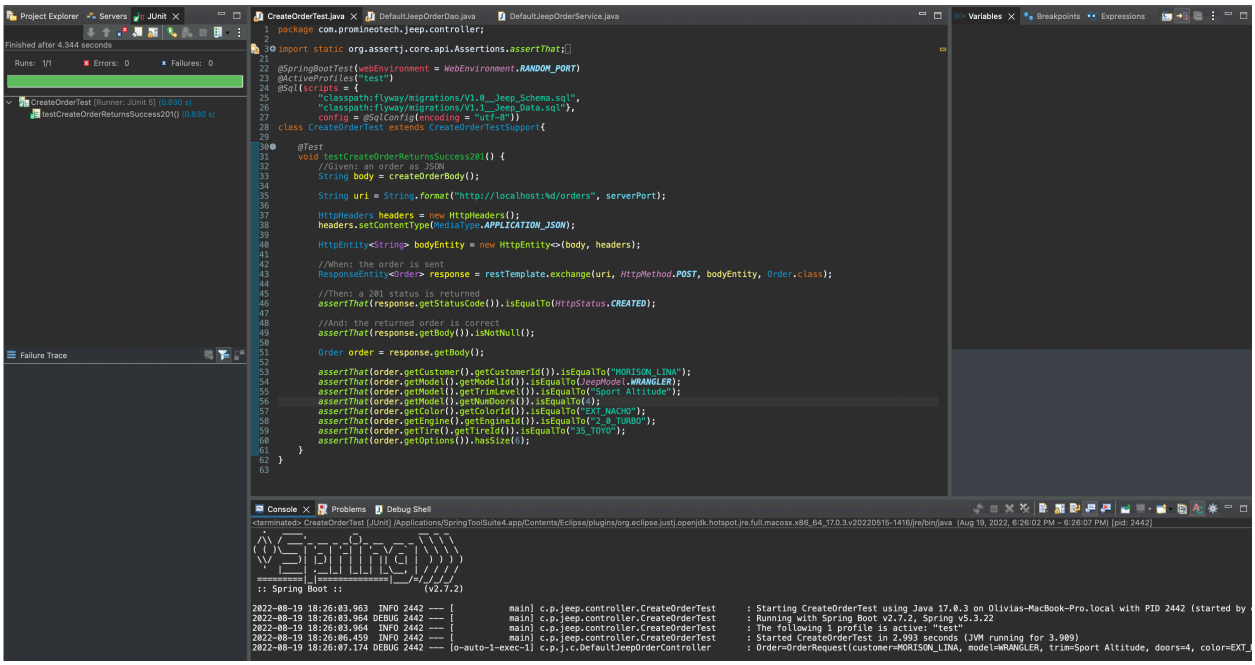
For each option in the `Options` list, call the supplied `generateInsertSql` method passing the parameters `option` and order primary key (`orderPK`). Call the `update` method on the `NamedParameterJdbcTemplate` object.

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.

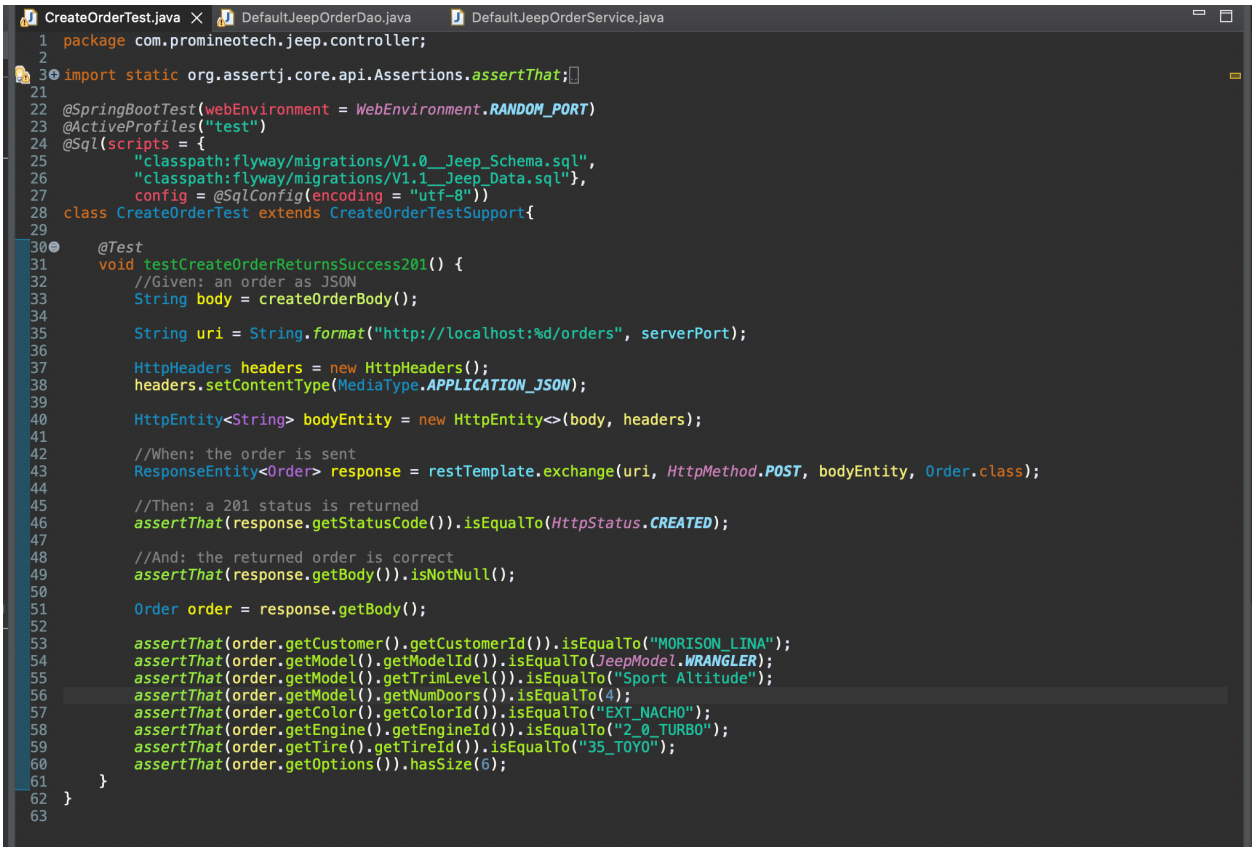
v) Produce a screenshot of the saveOrder method.

```
40    @Override
41    public Order saveOrder(Customer customer, Jeep jeep, Color color,
42            Engine engine, Tire tire, BigDecimal price, List<Option> options) {
43
44        SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
45
46        KeyHolder keyHolder = new GeneratedKeyHolder();
47        jdbcTemplate.update(params.sql, params.source, keyHolder);
48
49        Long orderPK = keyHolder.getKey().longValue();
50
51        saveOptions(options, orderPK);
52
53        // @formatter:off
54        return Order.builder()
55                .orderPK(orderPK)
56                .customer(customer)
57                .model(jeep)
58                .color(color)
59                .engine(engine)
60                .tire(tire)
61                .options(options)
62                .price(price)
63                .build();
64        // @formatter:on
65    }
66
```

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.


Project Explorer
Servers
JUnit
Finished after 4.344 seconds
Runs: 1/1
Errors: 0
Failures: 0
CreateOrderTest [Runner: JUnit 5] (0.830 s)
testCreateOrderReturnsSuccess201() (0.830 s)
Failure Trace
CreateOrderTest.java
DefaultJeepOrderDao.java
DefaultJeepOrderService.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 extends CreateOrderTestSupport{
@Test
void testCreateOrderReturnsSuccess201() {
//Given: an order as JSON
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);
//When: the order is sent
ResponseEntity<Order> response = restTemplate.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);
}
}
Variables
Breakpoints
Expressions
Console
Problems
Debug Shell
<terminated> CreateOrderTest [JUnit] /Applications/SpringToolSuite4.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.3.v20220515-1416/jre/bin/java (Aug 19, 2022, 6:26:02 PM – 6:26:07 PM) [pid: 2442]
:: Spring Boot ::
(v2.7.2)
2022-08-19 18:26:03.963 INFO 2442 --- [main] c.p.jeep.controller.CreateOrderTest : Starting CreateOrderTest using Java 17.0.3 on Olivias-MacBook-Pro.local with PID 2442 (started by
2022-08-19 18:26:03.964 DEBUG 2442 --- [main] c.p.jeep.controller.CreateOrderTest : Running with Spring Boot v2.7.2, Spring v5.3.22
2022-08-19 18:26:03.964 INFO 2442 --- [main] c.p.jeep.controller.CreateOrderTest : The following 1 profile is active: "test"
2022-08-19 18:26:06.459 INFO 2442 --- [main] c.p.jeep.controller.CreateOrderTest : Started CreateOrderTest in 2.993 seconds (JVM running for 3.909)
2022-08-19 18:26:07.174 DEBUG 2442 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport Altitude, doors=4, color=EXT_

Screenshots of Code:



CreateOrderTest.java | DefaultJeepOrderDao.java | DefaultJeepOrderService.java

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

import java.math.BigDecimal;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Component;

import com.promineotech.jeep.entity.Color;
import com.promineotech.jeep.entity.Customer;
import com.promineotech.jeep.entity.Engine;
import com.promineotech.jeep.entity.FuelType;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.JeepModel;
import com.promineotech.jeep.entity.Option;
import com.promineotech.jeep.entity.OptionType;
import com.promineotech.jeep.entity.Order;
import com.promineotech.jeep.entity.Tire;

@Component
public class DefaultJeepOrderDao implements JeepOrderDao {

    @Autowired
      private NamedParameterJdbcTemplate jdbcTemplate;

    @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
    }

    /**
     *
     * @param options
     * @param orderPK
     */
     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

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

        return params;
      }
```

```
100
101     /**
102      *
103      * @param customer
104      * @param jeep
105      * @param color
106      * @param engine
107      * @param tire
108      * @param price
109      * @return
110      */
111     private SqlParams generateInsertSql(Customer customer, Jeep jeep, Color color,
112         Engine engine, Tire tire, BigDecimal price) {
113       // @formatter:off
114       String sql = ""
115           + "INSERT INTO orders ("
116           + "customer_fk, color_fk, engine_fk, tire_fk, model_fk, price"
117           + ") VALUES ("
118           + ":customer_fk, :color_fk, :engine_fk, :tire_fk, :model_fk, :price"
119           + ")";
120       // @formatter:on
121
122       SqlParams params = new SqlParams();
123
124       params.sql = sql;
125       params.source.addValue("customer_fk", customer.getCustomerPK());
126       params.source.addValue("color_fk", color.getColorPK());
127       params.source.addValue("engine_fk", engine.getEnginePK());
128       params.source.addValue("tire_fk", tire.getTirePK());
129       params.source.addValue("model_fk", jeep.getModelPK());
130       params.source.addValue("price", price);
131
132       return params;
133     }
134
135     /**
136      *
137      */
138     @Override
139     public List<Option> fetchOptions(List<String> optionIds) {
140       if (optionIds.isEmpty()) {
141         return new LinkedList<>();
142       }
143
144       Map<String, Object> params = new HashMap<>();
145
146       // @formatter:off
147       String sql = ""
148           + "SELECT * "
149           + "FROM options "
150           + "WHERE option_id IN(";
151       // @formatter:on
152
153       for (int index = 0; index < optionIds.size(); index++) {
154         String key = "option_" + index;
155         sql += ":" + key + ", ";
156         params.put(key, optionIds.get(index));
157       }
158
159       sql = sql.substring(0, sql.length() - 2);
160       sql += ")";
161
162       return jdbcTemplate.query(sql, params, new RowMapper<Option>() {
163         @Override
164         public Option mapRow(ResultSet rs, int rowNum) throws SQLException {
165           // @formatter:off
166           return Option.builder()
167               .category(OptionType.valueOf(rs.getString("category")))
168               .manufacturer(rs.getString("manufacturer"))
169               .name(rs.getString("name"))
170               .optionId(rs.getString("option_id"))
171               .optionPK(rs.getLong("option_pk"))
172               .price(rs.getBigDecimal("price"))
173               .build();
174           // @formatter:on
175         }
176       });
177     }
178
179     /**
180      *
181      */
182     @Override
183     public Optional<Customer> fetchCustomer(String customerId) {
184       // @formatter:off
185       String sql = ""
186           + "SELECT * "
187           + "FROM customers "
188           + "WHERE customer_id = :customer_id";
189       // @formatter:on
190
191       Map<String, Object> params = new HashMap<>();
192       params.put("customer_id", customerId);
193
194       return Optional.ofNullable(
195           jdbcTemplate.query(sql, params, new CustomerResultSetExtractor()));
196     }
197
198     /**
199      *
200      */
201     @Override
202     public Optional<Jeep> fetchModel(JeepModel model, String trim, int doors) {
203       // @formatter:off
204       String sql = ""
205           + "SELECT * "
206           + "FROM models "
207           + "WHERE model_id = :model_id "
208           + "AND trim_level = :trim_level "
209           + "AND num_doors = :num_doors";
210       // @formatter:on
211
212       Map<String, Object> params = new HashMap<>();
213       params.put("model_id", model.toString());
214       params.put("trim_level", trim);
215       params.put("num_doors", doors);
```

```
216
217           return Optional.ofNullable(
218               jdbcTemplate.query(sql, params, new ModelResultSetExtractor()));
219         }
220
221         /**
222          *
223          */
224         @Override
225         public Optional<Color> fetchColor(String colorId) {
226           // @formatter:off
227           String sql = ""
228               + "SELECT * "
229               + "FROM colors "
230               + "WHERE color_id = :color_id";
231           // @formatter:on
232
233           Map<String, Object> params = new HashMap<>();
234           params.put("color_id", colorId);
235
236           return Optional.ofNullable(
237               jdbcTemplate.query(sql, params, new ColorResultSetExtractor()));
238         }
239
240         /**
241          *
242          */
243         @Override
244         public Optional<Engine> fetchEngine(String engineId) {
245           // @formatter:off
246           String sql = ""
247               + "SELECT * "
248               + "FROM engines "
249               + "WHERE engine_id = :engine_id";
250           // @formatter:on
251
252           Map<String, Object> params = new HashMap<>();
253           params.put("engine_id", engineId);
254
255           return Optional.ofNullable(
256               jdbcTemplate.query(sql, params, new EngineResultSetExtractor()));
257         }
258
259         /**
260          *
261          */
262         @Override
263         public Optional<Tire> fetchTire(String tireId) {
264           // @formatter:off
265           String sql = ""
266               + "SELECT * "
267               + "FROM tires "
268               + "WHERE tire_id = :tire_id";
269           // @formatter:on
270
271           Map<String, Object> params = new HashMap<>();
272           params.put("tire_id", tireId);
273
```

```
274           return Optional.ofNullable(
275               jdbcTemplate.query(sql, params, new TireResultSetExtractor()));
276         }
277
278         /**
279          *
280          * @author Promineo
281          *
282          */
283         class TireResultSetExtractor implements ResultSetExtractor<Tire> {
284           @Override
285           public Tire extractData(ResultSet rs) throws SQLException {
286             rs.next();
287
288             // @formatter:off
289             return Tire.builder()
290                 .manufacturer(rs.getString("manufacturer"))
291                 .price(rs.getBigDecimal("price"))
292                 .tireId(rs.getString("tire_id"))
293                 .tirePK(rs.getLong("tire_pk"))
294                 .tireSize(rs.getString("tire_size"))
295                 .warrantyMiles(rs.getInt("warranty_miles"))
296                 .build();
297             // @formatter:on
298           }
299         }
300
301         /**
302          *
303          * @author Promineo
304          *
305          */
306         class EngineResultSetExtractor implements ResultSetExtractor<Engine> {
307           @Override
308           public Engine extractData(ResultSet rs) throws SQLException {
309             rs.next();
310
311             // @formatter:off
312             return Engine.builder()
313                 .description(rs.getString("description"))
314                 .engineId(rs.getString("engine_id"))
315                 .enginePK(rs.getLong("engine_pk"))
316                 .fuelType(FuelType.valueOf(rs.getString("fuel_type")))
317                 .hasStartStop(rs.getBoolean("has_start_stop"))
318                 .mpgCity(rs.getFloat("mpg_city"))
319                 .mpgHwy(rs.getFloat("mpg_hwy"))
320                 .name(rs.getString("name"))
321                 .price(rs.getBigDecimal("price"))
322                 .sizeInLiters(rs.getFloat("size_in_liters"))
323                 .build();
324             // @formatter:on
325           }
326         }
327
328         /**
329          *
330          * @author Promineo
331          *
```

```
332         */
333       class ColorResultSetExtractor implements ResultSetExtractor<Color> {
334         @Override
335         public Color extractData(ResultSet rs) throws SQLException {
336           rs.next();
337
338           // @formatter:off
339           return Color.builder()
340               .color(rs.getString("color"))
341               .colorId(rs.getString("color_id"))
342               .colorPK(rs.getLong("color_pk"))
343               .isExterior(rs.getBoolean("is_exterior"))
344               .price(rs.getBigDecimal("price"))
345               .build();
346           // @formatter:on
347         }
348       }
349
350       /**
351        *
352        * @author Prominea
353        *
354        */
355       class ModelResultSetExtractor implements ResultSetExtractor<Jeep> {
356         @Override
357         public Jeep extractData(ResultSet rs) throws SQLException {
358           rs.next();
359
360           // @formatter:off
361           return Jeep.builder()
362               .basePrice(rs.getBigDecimal("base_price"))
363               .modelId(JeepModel.valueOf(rs.getString("model_id")))
364               .modelPK(rs.getLong("model_pk"))
365               .numDoors(rs.getInt("num_doors"))
366               .trimLevel(rs.getString("trim_level"))
367               .wheelSize(rs.getInt("wheel_size"))
368               .build();
369           // @formatter:on
370         }
371       }
372
373       /**
374        *
375        * @author Prominea
376        *
377        */
378       class CustomerResultSetExtractor implements ResultSetExtractor<Customer> {
379         @Override
380         public Customer extractData(ResultSet rs) throws SQLException {
381           rs.next();
382
383           // @formatter:off
384           return Customer.builder()
385               .customerId(rs.getString("customer_id"))
386               .customerPK(rs.getLong("customer_pk"))
387               .firstName(rs.getString("first_name"))
388               .lastName(rs.getString("last_name"))
389               .phone(rs.getString("phone"))
390               .build();
391           // @formatter:on
392
393         }
394       }
395
396       class SqlParams {
397         String sql;
398         MapSqlParameterSource source = new MapSqlParameterSource();
399       }
400   }
```

CreateOrderTest.java | DefaultJeepOrderDao.java | DefaultJeepOrderService.java

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

import java.math.BigDecimal;
import java.util.List;
import java.util.NoSuchElementException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.promineotech.jeep.dao.JeepOrderDao;
import com.promineotech.jeep.entity.Color;
import com.promineotech.jeep.entity.Customer;
import com.promineotech.jeep.entity.Engine;
import com.promineotech.jeep.entity.Jeep;
import com.promineotech.jeep.entity.Option;
import com.promineotech.jeep.entity.Order;
import com.promineotech.jeep.entity.OrderRequest;
import com.promineotech.jeep.entity.Tire;

@Service
public class DefaultJeepOrderService implements JeepOrderService {

    @Autowired
    private JeepOrderDao jeepOrderDao;

    @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);
    }

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

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

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

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

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

CreateOrderTest.java | DefaultJeepOrderController.java

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

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

@RestController
@Slf4j
public class DefaultJeepOrderController implements JeepOrderController {

    @Autowired
    private JeepOrderService jeepOrderService;

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

```

CreateOrderTest.java | JeepOrderController.java

```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;
import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.servers.Server;

@Validated
@RequestMapping("/orders")
@OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
        @Server(url = "http://localhost:8080", description = "Local server.") })
public interface JeepOrderController {

    // @formatter:off
    @Operation(
        summary = "Create an order for a Jeep",
        description = "Returns the created Jeep",
        responses = {
            @ApiResponse(
                responseCode = "201",
                description = "The created Jeep is returned.",
                content = @Content(mediaType = "application/json",
                schema = @Schema(implementation = Order.class))),
            @ApiResponse(
                responseCode = "400",
                description = "The request parameters are invalid.",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "404",
                description = "A Jeep component was not found with the input criteria.",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "500",
                description = "An unplanned error accurred.",
                content = @Content(mediaType = "application/json"))
        },
        parameters = {
            @Parameter(name = "orderRequest",
                required = true,
                description = "The order as JSON")
        }
    )

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

```

CreateOrderTest.java | DefaultJeepOrderDao.java | JeepOrderDao.java

```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);
}
```

CreateOrderTest.java | JeepOrderService.java

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

import com.promineotech.jeep.entity.Order;

public interface JeepOrderService {

    Order createOrder(OrderRequest orderRequest);

}
```

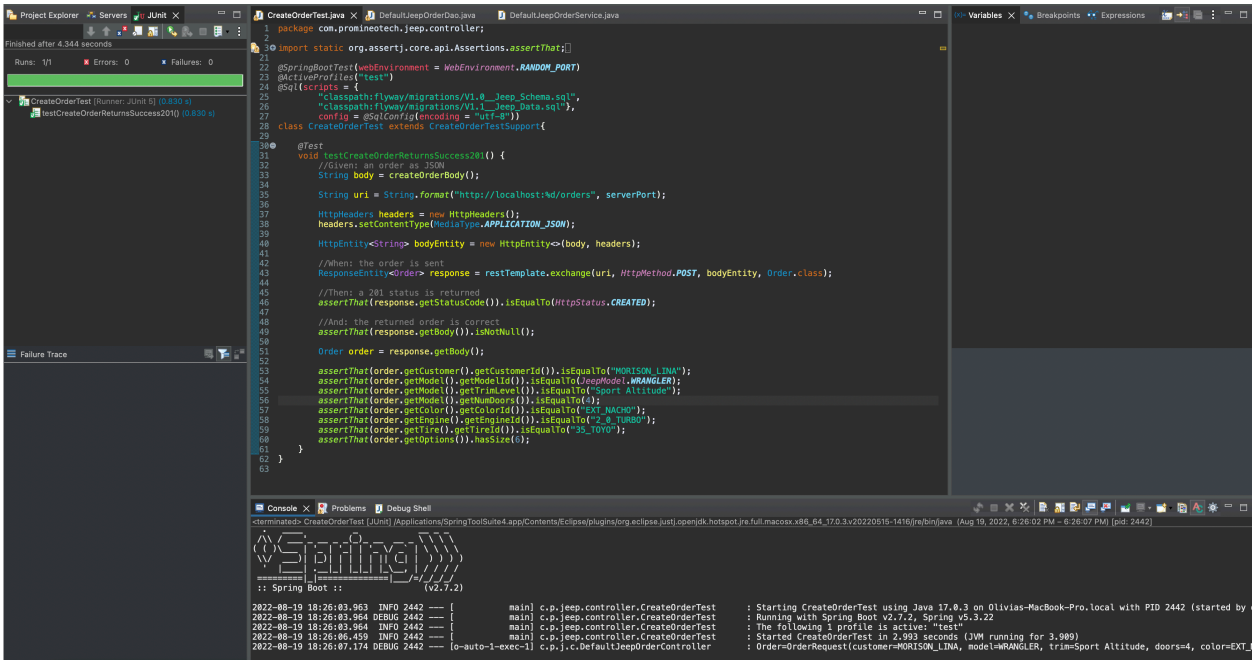
CreateOrderTest.java | BaseTest.java

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

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

abstract class BaseTest {
    @LocalServerPort
    protected int serverPort;

    @Autowired
    @Getter
    protected TestRestTemplate restTemplate;

    /**
     *
     * @return
     */
    protected String getBaseUriForJeeps() {
        return String.format("http://localhost:%d/jeeps", serverPort);
    }

    /**
     *
     * @return
     */
    protected String getBaseUriForOrders() {
        return String.format("http://localhost:%d/orders", serverPort);
    }
}
```

CreateOrderTest.java | CreateOrderTestSupport.java

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


public class CreateOrderTestSupport extends BaseTest{    //extends BaseTest to get the common methods

    /**
     *
     * @return
     */

    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
    }
}

```

Screenshots of Running Application:



URL to GitHub Repository:

https://github.com/OliGuzman/Spring-Boot-Web-API-Design