Web API Design with Spring Boot Week 4 Coding Assignment

Points possible: 70

Category	Criteria	% of Grade
Functionality	Does the code work?	25
Organization	Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear.	25
Creativity	Student solved the problems presented in the assignment using creativity and out of the box thinking.	25
Completeness	All requirements of the assignment are complete.	25

Instructions: In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed. Take screenshots of the code and of the running program (make sure to get screenshots of all required functionality) and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document, with your Java project code, to the repository. Add the URL for this week's repository to this document where instructed and submit this document to your instructor when complete.

Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon: 📹 You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.

Project Resources: https://github.com/promineotech/Spring-Boot-Course-Student-Resources

Coding Steps:

For this week's homework you need to copy source code from the supplied resources.

For this week's homework you need to copy source code from the Source folder in the supplied resources. Wait until the instructions tell you to copy the resources or you will get errors.

1) Select some options for a Jeep order:

   a) Use the `data.sql` file or the jeep database tables to select options for a Jeep order. Select any one of each of the following for the order:

      i) color

         EXT_SNAZZBERRY', 'Snazzberry Pearl Coat', 245.00, 1

      ii) customer

         'ROTH_GARTH', 'Garth', 'Roth', '312.753.9994'

      iii) engine

         '2_0_TURBO', 2.0, '2.0L I4 DOHC DI Turbo Engine with Start Stop', 'GASOLINE', 22, 24, 1, 'The 2.0L Four-Cylinder DOHC Turbo engine with Engine Stop/Start (ESS) Technology provides plenty of power for on or off-road driving. ESS automatically shuts the engine off at full stops, then re-engages for takeoff. The technology is engineered to help deliver an efficient performance, and can be disabled with the push of a button', 0.00);

      iv) model

         'WRANGLER', 'Rubicon', 4, 17, 42620.00

      v) tire(s)

         '255_GOODYEAR', '255/70R18 All-Terrain Tires', 'Goodyear', 0.00, 40000

   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!

      'DOOR_WARRIOR_MIRROR_MT', 'DOOR', 'Warrior Products', 'Tube Door Mirror Mounts', 61.29

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.

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.

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

a) Create a method in the interface to create an order (`createOrder`). It should return an object of type `Order` (see below). It should accept a single parameter of type `OrderRequest` as described in the video. Make sure it accepts an `HTTP POST` request and returns a status code of 201 (created).

b) Add the `@RequestBody` annotation to the orderRequest parameter. Make sure to add the `RequestBody` annotation from the `org.springframework.web.bind.annotation` package.

c) Produce a screenshot of the finished `JeepOrderController` interface showing no compile errors.

4) Create a class that implements `JeepOrderController` named `DefaultJeepOrderController`.

   a) Add `@RestController` as a class-level annotation.

   b) Add a log line to the implementing controller method showing the input request body (orderRequest)

   c) Run the test to show a red status bar. Produce a screenshot that shows the test method, the log line, and the red JUnit status bar.

5) Find the Maven dependency `spring-boot-starter-validation` by looking it up at https://mvnrepository.com/. Add this repository to the project POM file (pom.xml).

6) Add the class-level annotation `@Validated` to the `JeepOrderController` interface.

7) Add Bean Validation annotations to the `OrderRequest` class as shown in the video.

   a) Use these annotations for String types:

      i) `@NotNull`

      ii) `@Length(max = 30)`

      iii) `@Pattern(regexp = "[\\w\\s]*")`

   b) Use these annotations for integer types:

      i) `@Positive`

      ii) `@Min(2)`

      iii) `@Max(4)`

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

d) Add validation to the list element (type String) by adding the validation annotations *inside* the generic definition. So, to add the String validation to the options, you would do this:

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

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

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

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

a) Inject the interface into the order controller implementation class.

b) Add the `@Service` annotation to the service implementation class.

c) Create the `createOrder` method in the interface and implementing service. The method signature should look like this:

```
Order createOrder(OrderRequest orderRequest);
```

d) Call the `createOrder` method from the controller and return the value returned by the service.

e) Add a log line in the createOrder method and log the `orderRequest` parameter.

f) Run the test CreateOrderTest again. Produce a screenshot showing that the `createOrder` method in the service was called in the service class.

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) Replace the entire contents of `DefaultJeepOrderDao.java` with the source found in `DefaultJeepOrderDao.source`. The source file is found in the Source folder of the supplied

project resources. After this step you will see errors in `DefaultJeepOrderDao`. This will be fixed shortly.

13) 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`.

14) 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.

15) 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.

16) 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 method from the order service. Produce a screenshot of the service method.

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

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

        ii) Call the `update` method on the `NamedParameterJdbcTemplate` object, passing in a `KeyHolder` object as shown in the video. Create the `KeyHolder` like this:

```
KeyHolder keyHolder = new GeneratedKeyHolder();
```

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

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

```
private void saveOptions(List<Option> options, Long orderPK)
```

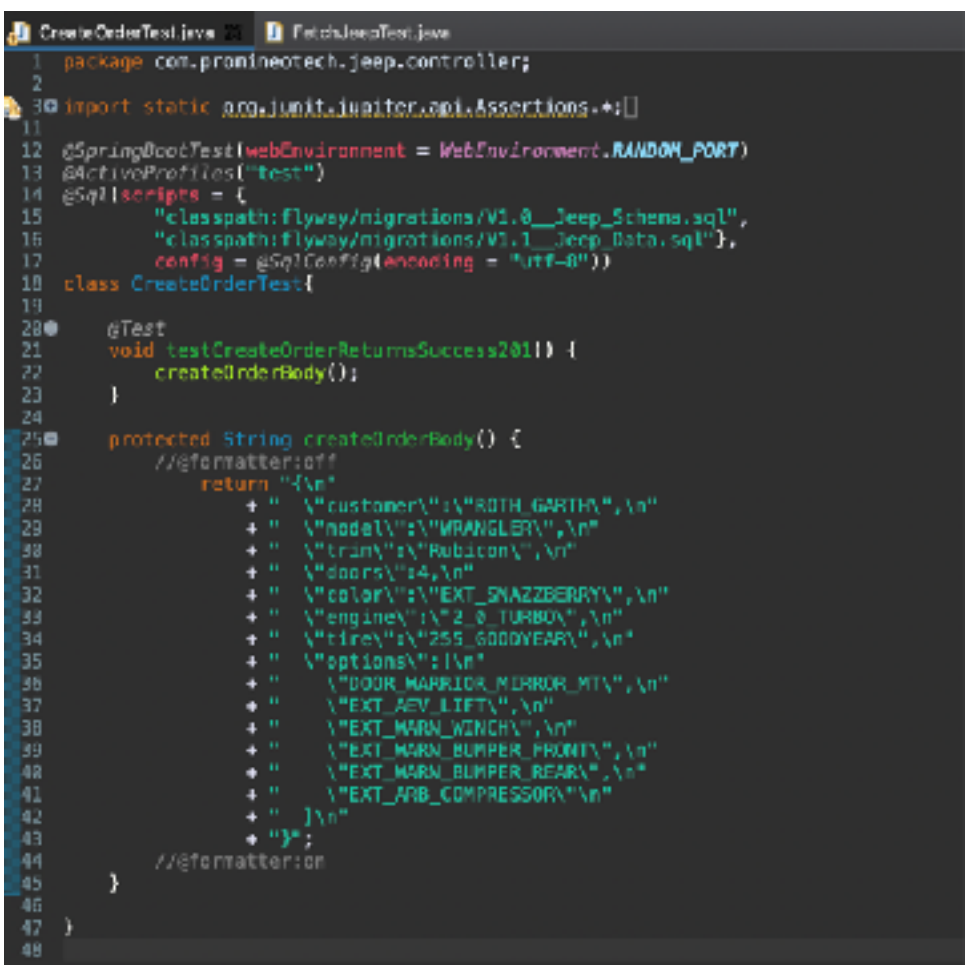
For each option in the `Options` list, call the supplied `generateInsertSql` method passing the parameters `option` and order primary key (`orderPK`). Call the `update` method on the `NamedParameterJdbcTemplate` object.

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

v) Produce a screenshot of the saveOrder method.

c) Run the integration test in `CreateOrderTest`. Produce a screenshot of the test method that shows the green JUnit status bar, the console output, and the test class.

Screenshots of Code/Running Application:



```
CreateOrderTest.java    FetchJeepsTest.java
1  package com.promineotech.jeep.controller;
2
3  import static org.junit.jupiter.api.Assertions.*;
11
12 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
13 @ActiveProfiles("test")
14 @Sql(scripts = {
15         "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
16         "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
17         config = @SqlConfig(encoding = "utf-8"))
18 class CreateOrderTest {
19
20     @Test
21     void testCreateOrderReturnsSuccess201() {
22         createOrderBody();
23     }
24
25     protected String createOrderBody() {
26         //@formatter:off
27             return "{\n"
28                 + "  \"customer\":\"ROTH_GARTH\",\n"
29                 + "  \"model\":\"WRANGLER\",\n"
30                 + "  \"trim\":\"Rubicon\",\n"
31                 + "  \"doors\":4,\n"
32                 + "  \"color\":\"EXT_SMAZZBERRY\",\n"
33                 + "  \"engine\":\"2_0_TURBO\",\n"
34                 + "  \"tire\":\"255_GOODYEAR\",\n"
35                 + "  \"options\":[\n"
36                 + "    \"DOOR_WARRIOR_MIRROR_MT\",\n"
37                 + "    \"EXT_AEV_LIFT\",\n"
38                 + "    \"EXT_WARN_WINCH\",\n"
39                 + "    \"EXT_WARN_BUMPER_FRONT\",\n"
40                 + "    \"EXT_WARN_BUMPER_REAR\",\n"
41                 + "    \"EXT_ARB_COMPRESSOR\"\n"
42                 + "  ]\n"
43                 + "}";
44         //@formatter:on
45     }
46
47 }
48
```

CreateOrderTest.java | FetchJeepTest.java

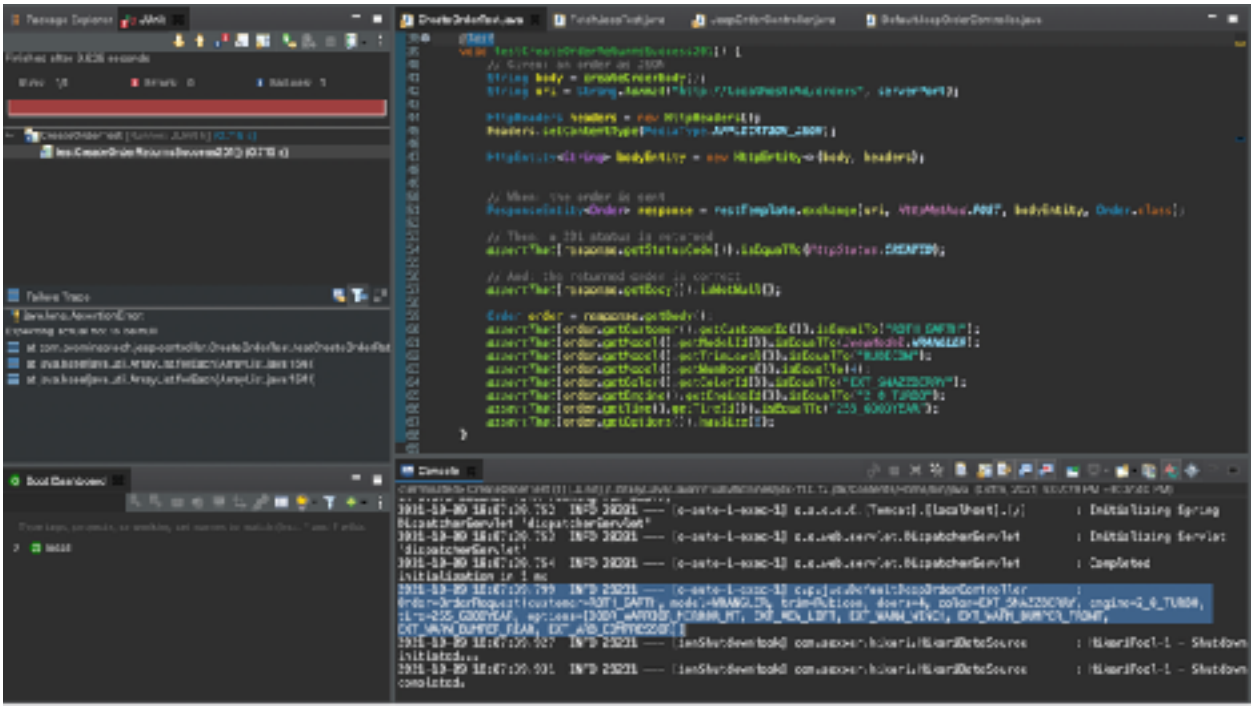
```java
package com.protineotech.jeep.controller;

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{

    @Autowired
    private TestRestTemplate restTemplate;

    @LocalServerPort
    private int serverPort;

    @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("ROTH_GARTH");
        assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
        assertThat(order.getModel().getTrimLevel()).isEqualTo("RUBICON");
        assertThat(order.getModel().getNumDoors()).isEqualTo(4);
        assertThat(order.getColor().getColorId()).isEqualTo("EXT_SNAZZBERRY");
        assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
        assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
        assertThat(order.getOptions()).hasSize(6);
    }
```

CreateOrderTest.java | FetchJeepTest.java

```java
        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("ROTH_GARTH");
        assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
        assertThat(order.getModel().getTrimLevel()).isEqualTo("RUBICON");
        assertThat(order.getModel().getNumDoors()).isEqualTo(4);
        assertThat(order.getColor().getColorId()).isEqualTo("EXT_SNAZZBERRY");
        assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
        assertThat(order.getTire().getTireId()).isEqualTo("255_GOODYEAR");
        assertThat(order.getOptions()).hasSize(6);
    }

    protected String createOrderBody() {
        //@formatter:off
            return "{\n"
                + "  \"customer\":\"ROTH_GARTH\",\n"
                + "  \"model\":\"WRANGLER\",\n"
                + "  \"trim\":\"Rubicon\",\n"
                + "  \"doors\":4,\n"
                + "  \"color\":\"EXT_SNAZZBERRY\",\n"
                + "  \"engine\":\"2_0_TURBO\",\n"
                + "  \"tire\":\"255_GOODYEAR\",\n"
                + "  \"options\":[\n"
                + "    \"DOOR_WARRIOR_MIRROR_MT\",\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
    }

}
```

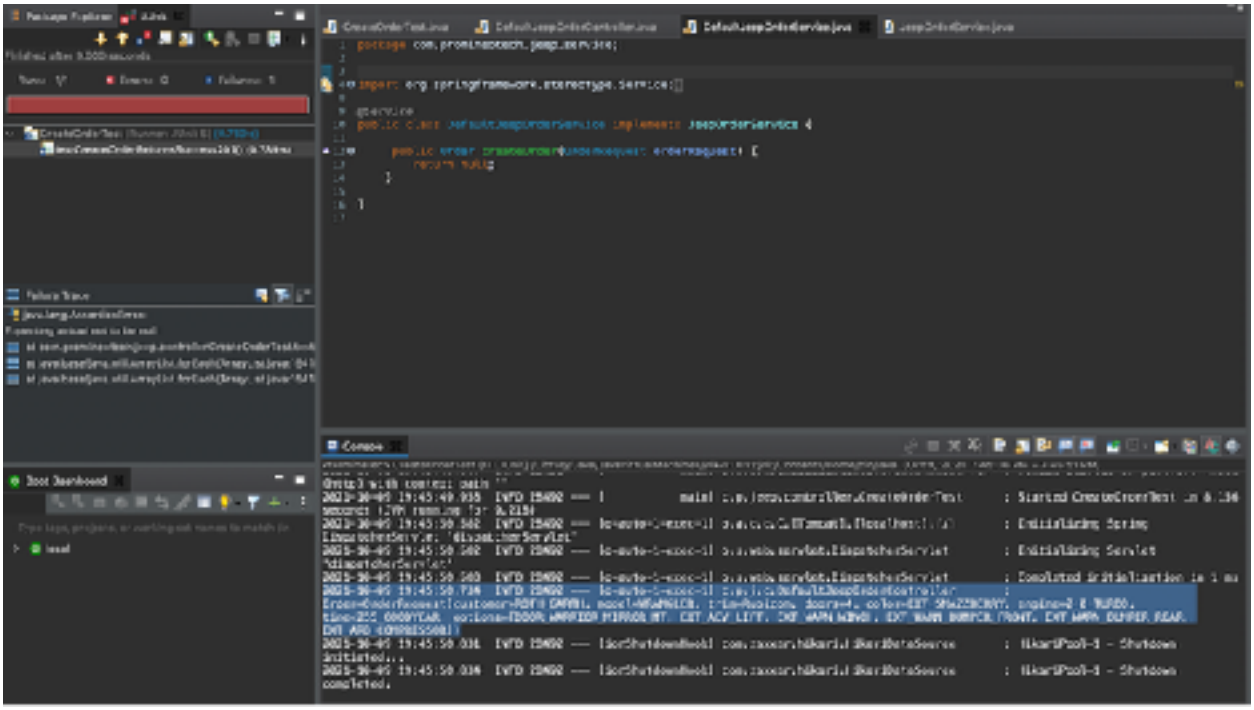
CreateOrderTest.java | FetchJeepTest.java | JeepOrderController.java

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

import java.util.List;

@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 occurred.",
                        content = @Content(
                                mediaType = "application/json"))
            },
            parameters = {
                @Parameter(
                        name = "orderRequest",
                        required = true,
                        description = "The order as JSON")
            }
        )
        @PostMapping
        @ResponseStatus(code = HttpStatus.CREATED)
        Order createOrder( @RequestBody OrderRequest orderRequest);
        // @formatter:on
}
```



CreateOrderTest.java | JeepOrderController.java | *OrderRequest.java | Console

```java
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 tire;

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



```java
DefaultJeepOrderService.java   JeepOrderDao.java   *DefaultJeepOrderDao.java

package com.promineotech.jeep.service;

import java.math.BigDecimal;

@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);
    }
```

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

import java.math.BigDecimal;

@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<Options> 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
    }
```

```java
    @Test
    void testCreateOrderReturnsSuccessGET() {
        // Given: an order as JSON
        String body = createOrderBody();
        String url = 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(url, HttpMethod.POST, bodyEntity, Order.class);

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

        // And: the returned order is correct
        assertNotNull(response.getBody().shortValue());

        order order = response.getBody();
        assertNotNull(order.getCustomer().getLastName().isEqualTo("John_Lennon");
        assertNotNull(order.getModel().getModelCode().isEqualTo("XR_MAXIMUM");
        assertNotNull(order.getModel().getTrimLevel().isEqualTo("Rubicon");
        assertNotNull(order.getModel().getMsrp().isEqualTo(44);
        assertNotNull(order.getColor().getColorCode().isEqualTo("EX_WHITEHOUSE");
        assertNotNull(order.getEngine().getEngineCode().isEqualTo("J_8_TURBO");
        assertNotNull(order.getTire().getTireCode().isEqualTo("255_GOODYEAR");
        assertNotNull(order.getOption().toArray());
    }
```

```
Console
seconds: 12VM running for 9.916)
2021-10-05 22:21:53.889  INFO 19034 ---  [main] o.s.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
DispatcherServlet 'dispatcherServlet'
2021-10-05 22:21:53.889  INFO 19034 ---  [main] o.s.web.servlet.DispatcherServlet        : Initializing Servlet
'dispatcherServlet'
2021-10-05 22:21:53.918  INFO 19034 ---  [main] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2021-10-05 22:21:53.941  INFO 19034 ---  [main] o.p.j.c.DefaultJeepOrderController       :
OrderCreateRequest(customer=KOTH_CAVEY, model=WRANGLER, trim=Rubicon, exterior=, color=EXT_INFIBEITY, engine=J_8_TURBO,
tire=255_GOODYEAR, options=[DOOR_WARMING_MIRROR_KT, EXT_AEV_LIFT, EXT_BARA_WINCH, EXT_WARN_BUMPER_FRONT, EXT_BARA_BUMPER_REAR,
EXT_BOB_FENDERFLARE])
2021-10-05 22:21:54.144  INFO 19034 --- [SpringShutdownHook] com.zaxxer.hikari.HikariDataSource       : Hikari_Real-1 - Shutdown
initiated...
2021-10-05 22:21:54.147  INFO 19034 --- [SpringShutdownHook] com.zaxxer.hikari.HikariDataSource       : Hikari_Real-1 - Shutdown completed.
```

URL to GitHub Repository: