

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.
- 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.
 - Create a test method (annotated with `@Test`) named `testCreateOrderReturnsSuccess201`.
 - 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": "MORTISON_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.

```
65    }
66
67    String createOrderBody() {
68        return "{\n"
69            + "  \"customer\":\"STERN_TORO\",\\n"
70            + "  \"model\":\"COMPASS\",\\n"
71            + "  \"trim\":\"TRAILHAWK\",\\n"
72            + "  \"doors\":4,\\n"
73            + "  \"color\":\"OLIVE_GREEN\",\\n"
74            + "  \"engine\":\"3_0_DIESEL\",\\n"
75            + "  \"tire\":\"35_TOYO\",\\n"
76            + "  \"options\":[\n"
77            + "    \"DOOR_QUAD_4\",\\n"
78            + "    \"EXT_QUAD_ALUM_FRONT\",\\n"
79            + "    \"EXT_WARN_WINCH\",\\n"
80            + "    \"EXT_WARN_BUMPER_FRONT\",\\n"
81            + "    \"EXT_WARN_BUMPER_REAR\",\\n"
82            + "    \"EXT_ARB_COMPRESSOR\"\n"
83            + "  ]\\n"
84            + "}\\n"
85            + """;
86    }
87 }
```

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();
```

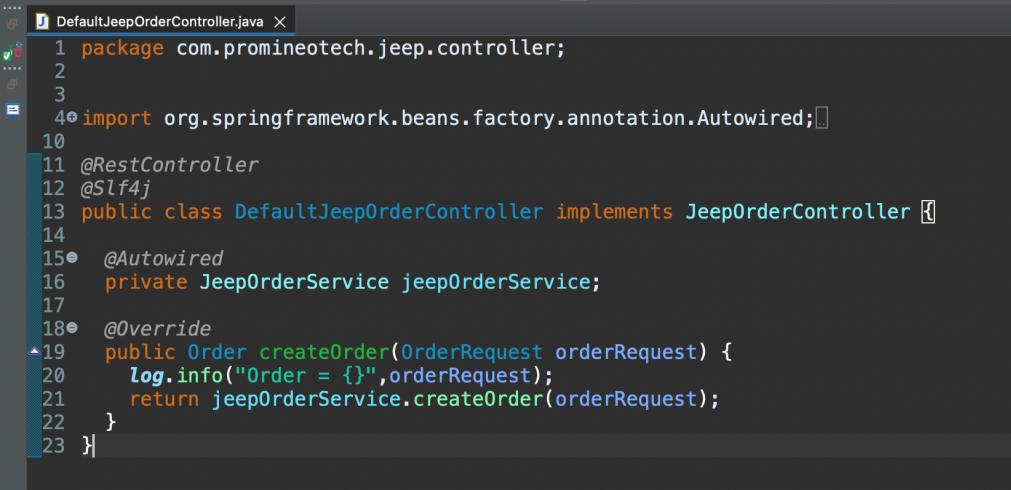
```
Order order = response.getBody();
assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
assertThat(order.getModel().getNumDoors()).isEqualTo(4);
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. 

```
41
42• @Test
43 void testCreateOrderReturnsSuccess201() {
44
45     String body = createOrderBody();
46     String uri = String.format("http://localhost:%d/orders", serverPort);
47     HttpHeaders headers = new HttpHeaders();
48     headers.setContentType(MediaType.APPLICATION_JSON);
49     HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
50     ResponseEntity<Order> response = restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, Order.class);
51
52     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
53
54     assertThat(response.getBody()).isNotNull();
55
56     Order order = response.getBody();
57     assertThat(order.getCustomer().getCustomerId()).isEqualTo("STERN_TORO");
58     assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.COMPASS);
59     assertThat(order.getModel().getTrimLevel()).isEqualTo("Trailhawk");
60     assertThat(order.getModel().getNumDoors()).isEqualTo(4);
61     assertThat(order.getColor().getColorId()).isEqualTo("OLIVE_GREEN");
62     assertThat(order.getEngine().getEngineId()).isEqualTo("3_0_DIESEL");
63     assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
64     assertThat(order.getOptions()).hasSize(6);
65 }
66 }
```

- 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



```
1 package com.promineotech.jeep.controller;
2
3
4 import org.springframework.beans.factory.annotation.Autowired;
5
6 @RestController
7 @Slf4j
8 public class DefaultJeepOrderController implements JeepOrderController {
9
10    @Autowired
11    private JeepOrderService jeepOrderService;
12
13    @Override
14    public Order createOrder(OrderRequest orderRequest) {
15        log.info("Order = {}", orderRequest);
16        return jeepOrderService.createOrder(orderRequest);
17    }
18}
19
20
21
22
23 }
```

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.  - Followed along with videos

The screenshot shows an IDE interface with three main panes:

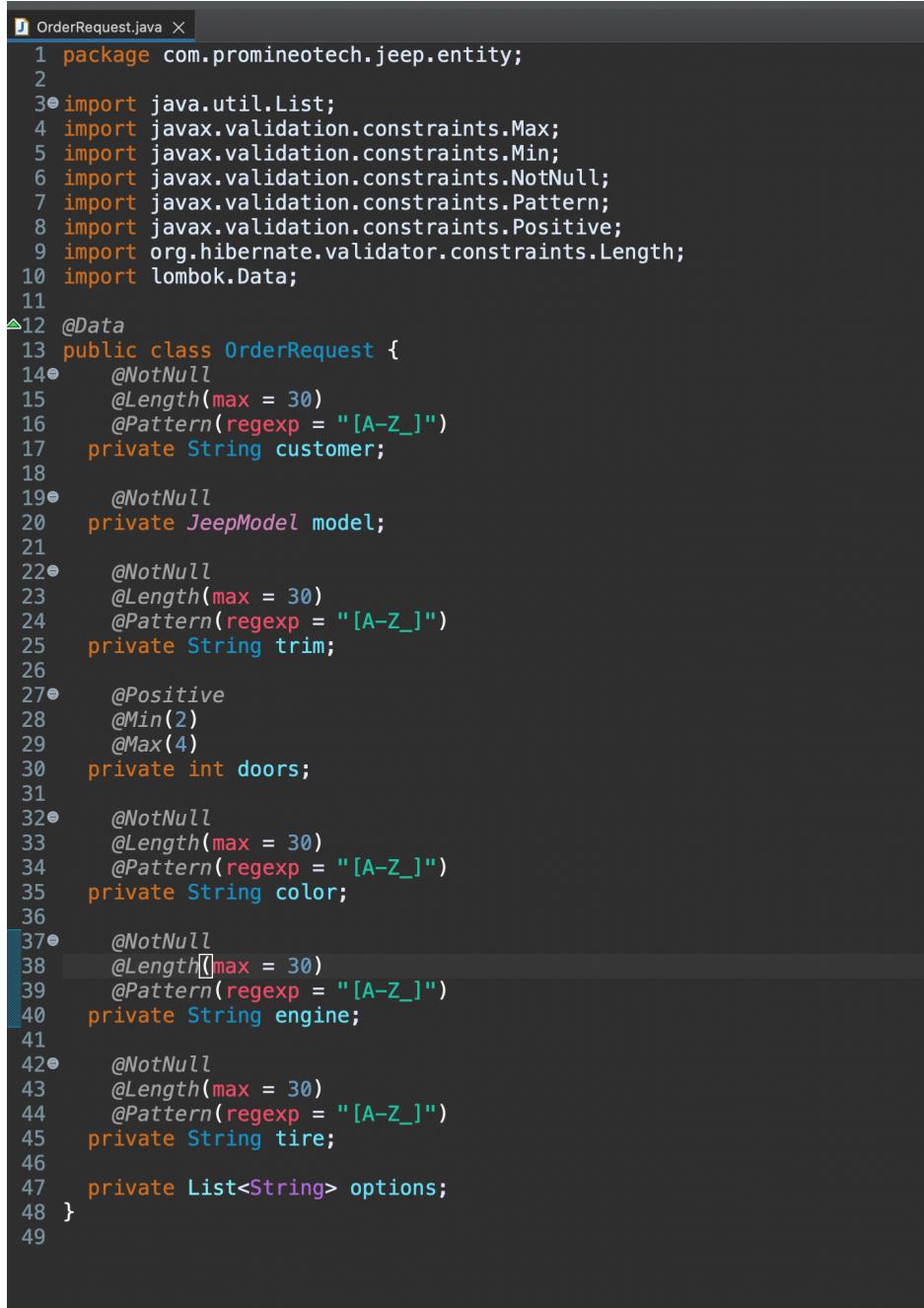
- JUnit X**: Shows a green bar indicating the test run was successful.
- Console X**: Displays the Spring Boot application's startup logs. It includes a decorative ASCII art logo at the top followed by several INFO-level log entries from 2022-10-12 22:06:03.223 to 2022-10-12 22:06:30.009.
- DefaultJeepOrderController.java**: Shows the Java code for the controller. It includes imports, annotations (@RestController, @Autowired), and a method implementation that logs the received OrderRequest and delegates to the service.

- 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. 

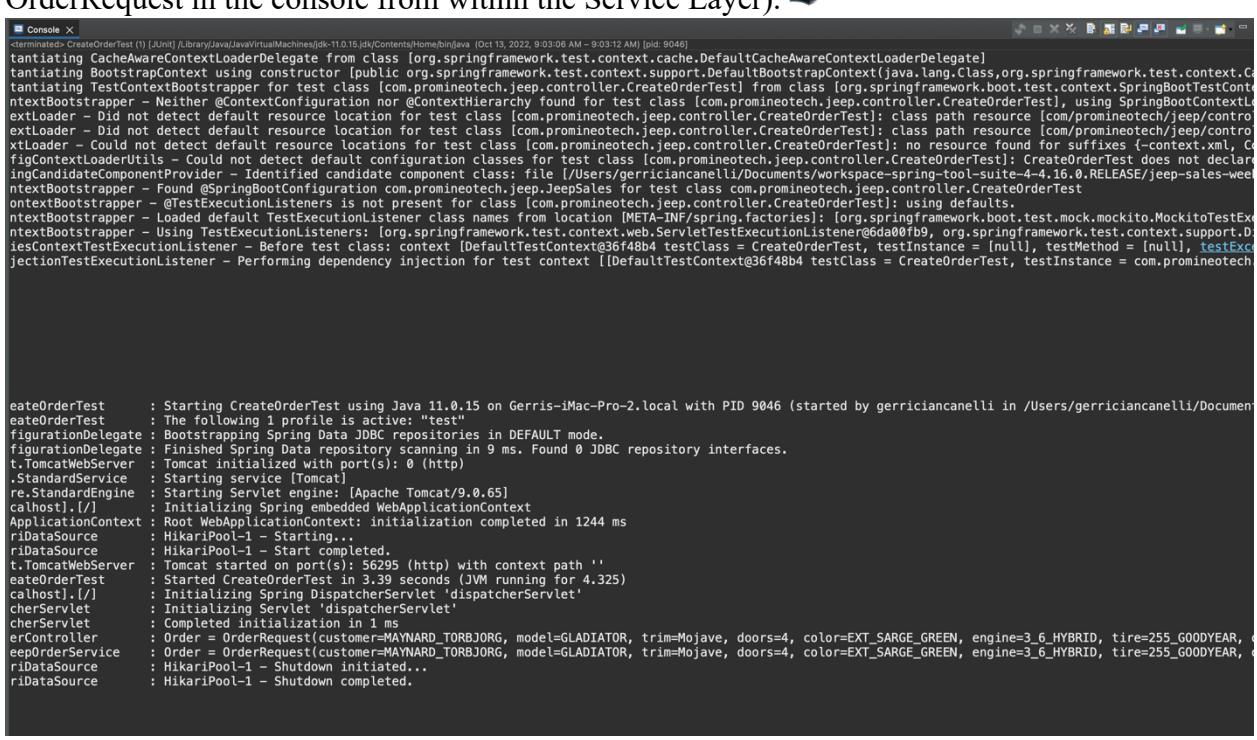


```
OrderRequest.java X
1 package com.promineotech.jeep.entity;
2
3 import java.util.List;
4 import javax.validation.constraints.Max;
5 import javax.validation.constraints.Min;
6 import javax.validation.constraints.NotNull;
7 import javax.validation.constraints.Pattern;
8 import javax.validation.constraints.Positive;
9 import org.hibernate.validator.constraints.Length;
10 import lombok.Data;
11
12 @Data
13 public class OrderRequest {
14     @NotNull
15     @Length(max = 30)
16     @Pattern(regexp = "[A-Z_]")
17     private String customer;
18
19     @NotNull
20     private JeepModel model;
21
22     @NotNull
23     @Length(max = 30)
24     @Pattern(regexp = "[A-Z_]")
25     private String trim;
26
27     @Positive
28     @Min(2)
29     @Max(4)
30     private int doors;
31
32     @NotNull
33     @Length(max = 30)
34     @Pattern(regexp = "[A-Z_]")
35     private String color;
36
37     @NotNull
38     @Length(max = 30)
39     @Pattern(regexp = "[A-Z_]")
40     private String engine;
41
42     @NotNull
43     @Length(max = 30)
44     @Pattern(regexp = "[A-Z_]")
45     private String tire;
46
47     private List<String> options;
48 }
49
```

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

The screenshot shows a Java IDE interface with two code files:

- JeepOrderService.java** (Top Tab):

```

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

```

20
21     @Transactional
22     public Order createOrder(OrderRequest orderRequest) {
23         log.info("Order = {}",orderRequest);
24
25         Customer customer = getCustomer(orderRequest);
26         Jeep jeep = getModel(orderRequest);
27         Color color = getColor(orderRequest);
28         Engine engine = getEngine(orderRequest);
29         Tire tire = getTire (orderRequest);
30         List<Option> options = getOption(orderRequest);
31
32         BigDecimal price =
33             jeep.getBasePrice()
34             .add(color.getPrice())
35             .add(engine.getPrice())
36             .add(tire.getPrice());
37
38         for(Option option : options) {
39             price = price.add(option.getPrice());
40         }
41
42         return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
43     }
44
45     /**
46      * ...
47     */
48 }
```

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

```

1 package com.promineotech.jeep.dao;
2
3+import java.math.BigDecimal;[]
4
5 public interface JeepOrderDao {
6     List<Option> fetchOptions(List<String> optionIds);
7     Optional<Customer> fetchCustomer(String customerId);
8     Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
9     Optional<Color> fetchColor(String colorId);
10    Optional<Engine> fetchEngine(String engineId);
11    Optional<Tire> fetchTire(String tireId);
12
13+@Override
14    Order saveOrder(
15        Customer customer,
16        Jeep jeep,
17        Color color,
18        Engine engine,
19        Tire tire,
20        BigDecimal price,
21        List<Option> options);
22
23}
24
25
26
27
28
29
30
31
32 }
```

- 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.

```

1 package com.promineotech.jeep.controller;
2
3+import static org.assertj.core.api.Assertions.assertThat;
4
5
6 @Validated
7 @WebEnvironment(testProfile = "test")
8 @ActiveProfiles("test")
9 @SqlScripts = {
10     @SqlScript(path = "classpath:/migrations/V1__Jeep_Schema.sql"),
11     @SqlScript(path = "classpath:/migrations/V1__Jeep_Data.sql"),
12     config = @SqlConfig(encoding = "utf-8")
13 }
14
15 public class CreateOrderTest {
16
17     @LocalServerTest
18     private int serverPort;
19
20     @Autowired
21     private TestRestTemplate restTemplate;
22
23     @Test
24     void testCreateOrderReturnsSuccess201() {
25
26         String body = createOrderBody();
27         String url = String.format("http://localhost:%d/orders", serverPort);
28         HttpHeaders headers = new HttpHeaders();
29         headers.setAccept(Arrays.asList(APPLICATION_JSON));
30         HttpEntity<String> bodyEntity = new HttpEntity<String>(body, headers);
31         ResponseEntity<Order> response = restTemplate.exchange(url, HttpMethod.POST, bodyEntity, Order.class);
32
33         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
34
35         assertThat(response.getBody()).isNotNull();
36
37         Order order = response.getBody();
38         assertThat(order.getCustomerId()).isEqualTo("MAYNARD_TORBJORG");
39         assertThat(order.getModelId()).isEqualTo(JeepModel.GLADIATOR);
40         assertThat(order.getModelId().getTrimLevel()).isEqualTo("W/java");
41         assertThat(order.getColorId()).isNotNull();
42         assertThat(order.getColorId()).isInequalTo("EXT_SANG_GREN");
43     }
44
45     private String createOrderBody() {
46         String body = "{\n47             \"customer\": {\n48                 \"name\": \"MAYNARD_TORBJORG\"\n49             },\n50             \"model\": {\n51                 \"trimLevel\": \"W/java\"\n52             },\n53             \"color\": {\n54                 \"id\": \"EXT_SANG_GREN\"\n55             },\n56             \"tire\": {\n57                 \"size\": \"22x10\"\n58             },\n59             \"options\": [\n60                 {\n61                     \"id\": \"OPTION_1\"\n62                 }\n63             ],\n64             \"price\": 100000\n65         }";
66     }
67 }

```

The screenshot also shows the Java console output at the bottom, displaying Spring Boot startup logs and Tomcat service logs.

Screenshots of Code:

```
1 package com.promineotech.jeep.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 @Validated
27 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
28 @ActiveProfiles("test")
29 @Sql scripts = {
30     "classpath:flyway/migrations/V1.0_Jeep_Schema.sql",
31     "classpath:flyway/migrations/V1.1_Jeep_Data.sql"}, config = @SqlConfig(encoding = "utf-8"))
32
33
34 public class CreateOrderTest {
35
36     @LocalServerPort
37     private int serverPort;
38
39     @Autowired
40     private TestRestTemplate restTemplate;
41
42     @Test
43     void testCreateOrderReturnsSuccess201() {
44
45         String body = createOrderBody();
46         String uri = String.format("http://localhost:%d/orders", serverPort);
47         HttpHeaders headers = new HttpHeaders();
48         headers.setContentType(MediaType.APPLICATION_JSON);
49         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
50         ResponseEntity<Order> response = restTemplate.exchange(uri, HttpMethod.POST, bodyEntity, Order.class);
51
52         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
53
54         assertThat(response.getBody()).isNotNull();
55
56         Order order = response.getBody();
57         assertThat(order.getCustomerId()).isEqualTo("STERN_TORO");
58         assertThat(order.getModelId()).isEqualTo(JeepModel.COMPASS);
59         assertThat(order.getTrimLevel()).isEqualTo("Trailhawk");
60         assertThat(order.getNumDoors()).isEqualTo(4);
61         assertThat(order.getColorId()).isEqualTo("OLIVE_GREEN");
62         assertThat(order.getEngineId()).isEqualTo("3_0_DIESEL");
63         assertThat(order.getTireId()).isEqualTo("35_TOYO");
64         assertThat(order.getOptions()).hasSize(6);
65     }
66 }
```

```
67     String createOrderBody() {
68         //formatter:off
69         return "{\n"
70             + "    \"customer\": \"STERN_TORO\", \n"
71             + "    \"model\": \"COMPASS\", \n"
72             + "    \"trim\": \"TRAILHAWK\", \n"
73             + "    \"doors\": 4, \n"
74             + "    \"color\": \"OLIVE_GREEN\", \n"
75             + "    \"engine\": \"3_0_DIESEL\", \n"
76             + "    \"tire\": \"35_TOYO\", \n"
77             + "    \"options\": [\n"
78                 + "        \"DOOR_QUAD_4\", \n"
79                 + "        \"EXT_QUAD_ALUM_FRONT\", \n"
80                 + "        \"EXT_WARN_WINCH\", \n"
81                 + "        \"EXT_WARN_BUMPER_FRONT\", \n"
82                 + "        \"EXT_WARN_BUMPER_REAR\", \n"
83                 + "        \"EXT_ARCOMPRESSOR\"\n"
84             + "    ]\n"
85             + """;
86
87         //formatter:on
88     }
89 }
```

```
1 package com.promineotech.jeep.controller;
2
3 import org.springframework.http.HttpStatus;
4
5 @Validated
6 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
7     @Server(url = "http://localhost:8080", description = "Local server."))}
8
9 @RequestMapping("/orders")
10
11 public interface JeepOrderController {
12
13     @Operation(
14         summary = "Create an order for a Jeep",
15         description = "Retruns the created Jeep",
16         responses = {
17             @ApiResponse(responseCode = "201",
18                 description = "The created Jeep is returned",
19                 content = @Content(mediaType = "application/json",
20                     schema = @Schema(implementation = Jeep.class))),
21             @ApiResponse(responseCode = "400",
22                 description = "The request parameters are invalid",
23                 content = @Content(mediaType = "application/json")),
24             @ApiResponse(responseCode = "404",
25                 description = "A Jeep component was not found with the input criteria",
26                 content = @Content(mediaType = "application/json")),
27             @ApiResponse(responseCode = "500",
28                 description = "An unplanned error occurred",
29                 content = @Content(mediaType = "application/json"))
30         },
31         parameters = {
32             @Parameter(name = "orderRequest",
33                 required = true,
34                 description = "The order as JSON"),
35         }
36     )
37     @PostMapping
38     @ResponseStatus(code = HttpStatus.CREATED)
39     Order createOrder(@RequestBody OrderRequest orderRequest);
40 }
```

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

```
CreateOrderTest.java  JeepOrderController.java  DefaultJeepOrderController.java  jeep-sales-week16_videowork/pom.xml ×
1 <?xml version="1.0" encoding="UTF-8"?>
2+<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5
6@  <parent>
7   <groupId>org.springframework.boot</groupId>
8   <artifactId>spring-boot-starter-parent</artifactId>
9   <version>2.7.4</version>
10  <relativePath/> <!-- lookup parent from repository -->
11 </parent>
12
13 <groupId>com.promineotech</groupId>
14 <artifactId>jeep-sales</artifactId>
15 <version>0.0.1-SNAPSHOT</version>
16
17 <name>jeep-sales</name>
18 <description>Jeep Sales</description>
19
20@  <properties>
21   <java.version>11</java.version>
22 </properties>
23
24@  <dependencies>
25@    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-starter-web</artifactId>
28    </dependency>
29
30@    <dependency>
31      <groupId>org.springframework.boot</groupId>
32      <artifactId>spring-boot-devtools</artifactId>
33      <scope>runtime</scope>
34      <optional>true</optional>
35    </dependency>
36
37@    <dependency>
38      <groupId>org.projectlombok</groupId>
39      <artifactId>lombok</artifactId>
40      <optional>true</optional>
41    </dependency>
42    <!-- Bean validation =====-->
43@    <dependency>
44      <groupId>org.springframework.boot</groupId>
45      <artifactId>spring-boot-starter-validation</artifactId>
46
47    </dependency>
48
```

```
49  <!-- OpenAPI dependency =====-->
50@  <dependency>
51      <groupId>org.springdoc</groupId>
52      <artifactId>springdoc-openapi-ui</artifactId>
53      <version>1.6.11</version>
54  </dependency>
55
56
57@  <dependency>
58      <groupId>org.springframework.boot</groupId>
59      <artifactId>spring-boot-starter-test</artifactId>
60      <scope>test</scope>
61  </dependency>
62
63@  <dependency>
64      <groupId>org.springframework.boot</groupId>
65      <artifactId>spring-boot-starter-data-jdbc</artifactId>
66  </dependency>
67
68@  <dependency>
69      <groupId>mysql</groupId>
70      <artifactId>mysql-connector-java</artifactId>
71  </dependency>
72 <!-- dependencies for tests =====-->
73@<dependency>
74      <groupId>com.h2database</groupId>
75      <artifactId>h2</artifactId>
76      <scope>test</scope>
77 </dependency>
78
79  </dependencies>
80
81@  <build>
82@    <plugins>
83@      <plugin>
84          <groupId>org.springframework.boot</groupId>
85          <artifactId>spring-boot-maven-plugin</artifactId>
86@        <configuration>
87@          <excludes>
88@            <exclude>
89                <groupId>org.projectlombok</groupId>
90                <artifactId>lombok</artifactId>
91            </exclude>
92        </excludes>
93        </configuration>
94    </plugin>
95  </plugins>
96 </build>
97
98 </project>
```

Overview Dependencies Dependency Hierarchy Effective POM pom.xml

```
1 package com.promineotech.jeep.entity;
2
3 import java.util.List;
4
5 @Data
6 public class OrderRequest {
7     @NotNull
8     @Length(max = 30)
9     @Pattern(regexp = "[A-Z_]")
10    private String customer;
11
12    @NotNull
13    private JeepModel model;
14
15    @NotNull
16    @Length(max = 30)
17    @Pattern(regexp = "[A-Z_]")
18    private String trim;
19
20    @Positive
21    @Min(2)
22    @Max(4)
23    private int doors;
24
25    @NotNull
26    @Length(max = 30)
27    @Pattern(regexp = "[A-Z_]")
28    private String color;
29
30    @NotNull
31    @Length(max = 30)
32    @Pattern(regexp = "[A-Z_]")
33    private String engine;
34
35    @NotNull
36    @Length(max = 30)
37    @Pattern(regexp = "[A-Z_]")
38    private String tire;
39
40    private List<String> options;
41
42}
43
44
```

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

```
1 package com.promineotech.jeep.service;
2
3 import java.math.BigDecimal;
4
5
6 @Service
7 @Slf4j
8 public class DefaultJeepOrderService implements JeepOrderService {
9
10    @Autowired
11    private JeepOrderDao jeepOrderDao;
12
13    @Transactional
14    public Order createOrder(OrderRequest orderRequest) {
15        log.info("Order = {}", orderRequest);
16
17        Customer customer = getCustomer(orderRequest);
18        Jeep jeep = getModel(orderRequest);
19        Color color = getColor(orderRequest);
20        Engine engine = getEngine(orderRequest);
21        Tire tire = getTire(orderRequest);
22        List<Option> options = getOption(orderRequest);
23
24        BigDecimal price =
25            jeep.getBasePrice()
26                .add(color.getPrice())
27                .add(engine.getPrice())
28                .add(tire.getPrice());
29
30        for(Option option : options) {
31            price = price.add(option.getPrice());
32        }
33
34        return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
35    }
36
37    /**
38     * @param orderRequest
39     * @return
40     */
41    private List<Option> getOption(OrderRequest orderRequest) {
42        return jeepOrderDao.fetchOptions(orderRequest.getOptions());
43    }
44
45    /**
46     * @param orderRequest
47     * @return
48     */
49    private Tire getTire(OrderRequest orderRequest) {
50        return jeepOrderDao.fetchTire(orderRequest.getTire())
51            .orElseThrow(() -> new NoSuchElementException(
52                "Tire with ID=" + orderRequest.getTire() + " was not found"));
53    }
54
55}
```

```
71● /**
72  *
73  * @param orderRequest
74  * @return
75  */
76● private Engine getEngine(OrderRequest orderRequest) {
77    return jeepOrderDao.fetchEngine(orderRequest.getEngine())
78    .orElseThrow(() -> new NoSuchElementException(
79        "Engine with ID=" + orderRequest.getEngine() + " was not found"));
80 }
81
82● /**
83  *
84  * @param orderRequest
85  * @return
86  */
87● private Color getColor(OrderRequest orderRequest) {
88    return jeepOrderDao.fetchColor(orderRequest.getColor())
89    .orElseThrow(() -> new NoSuchElementException(
90        "Color with ID=" + orderRequest.getColor() + " was not found"));
91 }
92
93● /**
94  *
95  * @param orderRequest
96  * @return
97  */
98● private Jeep getModel(OrderRequest orderRequest) {
99    return jeepOrderDao
100       .fetchModel(orderRequest.getModel(), orderRequest.getTrim(),
101                  orderRequest.getDoors())
102       .orElseThrow(() -> new NoSuchElementException("Model with ID="
103                  + orderRequest.getModel() + ", trim=" + orderRequest.getTrim()
104                  + orderRequest.getDoors() + " was not found"));
105 }
106
107● /**
108  *
109  * @param orderRequest
110  * @return
111  */
112● private Customer getCustomer(OrderRequest orderRequest) {
113    return jeepOrderDao.fetchCustomer(orderRequest.getCustomer())
114    .orElseThrow(() -> new NoSuchElementException("Customer with ID="
115                  + orderRequest.getCustomer() + " was not found"));
116 }
117 }
```

The screenshot shows a Java code editor with the file `*JeepOrderDao.java` selected. The code defines a public interface `JeepOrderDao` with various methods for fetching options, customers, models, colors, engines, and tires, and a method for saving an order. The code uses Java 8's `Optional` and `List` types, and imports `java.math.BigDecimal`.

```
1 package com.promineotech.jeep.dao;
2
3+ import java.math.BigDecimal;..
4
5 public interface JeepOrderDao {
6     List<Option> fetchOptions(List<String> optionIds);
7     Optional<Customer> fetchCustomer(String customerId);
8     Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
9     Optional<Color> fetchColor(String colorId);
10    Optional<Engine> fetchEngine(String engineId);
11    Optional<Tire> fetchTire(String tireId);
12
13+   Order saveOrder(
14         Customer customer,
15         Jeep jeep,
16         Color color,
17         Engine engine,
18         Tire tire,
19         BigDecimal price,
20         List<Option> options);
21
22 }
23 }
```

```
1 package com.promineotech.jeep.dao;
2
3 import java.math.BigDecimal;
4
5 @Component
6 public class DefaultJeepOrderDao implements JeepOrderDao {
7     @Autowired
8     private NamedParameterJdbcTemplate jdbcTemplate;
9
10    @Override
11    public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
12                           BigDecimal price, List<Option> options) {
13        SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
14
15        KeyHolder keyHolder = new GeneratedKeyHolder();
16        jdbcTemplate.update(params.sql, params.source, keyHolder);
17
18        Long orderPK = keyHolder.getKey().longValue();
19        saveOptions(options, orderPK);
20
21        return Order.builder()
22            .orderPK(orderPK)
23            .customer(customer)
24            .model(jeep)
25            .color(color)
26            .engine(engine)
27            .tire(tire)
28            .options(options)
29            .price(price)
30            .build();
31    }
32
33    private void saveOptions(List<Option> options, Long orderPK) {
34        for(Option option : options) {
35            SqlParams params = generateInsertSql(option, orderPK);
36            jdbcTemplate.update(params.sql, params.source);
37        }
38    }
39
40    /**
41     * @param option
42     * @param orderPK
43     * @return
44     */
45    private SqlParams generateInsertSql(Option option, Long orderPK) {
46        SqlParams params = new SqlParams();
47
48        // @formatter:off
49        params.sql = """
50            + "INSERT INTO order_options ("
51            + "option_fk, order_fk"
52            + ") VALUES ("
53            + ":option_fk, :order_fk"
54            + ")";
55        // @formatter:on
56
57        params.source = new BeanPropertySqlParameterSource(option);
58
59        return params;
60    }
61}
```

```
 86     params.source.addValue("option_fk", option.getOptionPK());
 87     params.source.addValue("order_fk", orderPK);
 88
 89     return params;
 90 }
 91
 92 /**
 93 *
 94 * @param customer
 95 * @param jeep
 96 * @param color
 97 * @param engine
 98 * @param tire
 99 * @param price
100 * @return
101 */
102 private SqlParams generateInsertSql(Customer customer, Jeep jeep, Color color,
103                                     Engine engine, Tire tire, BigDecimal price) {
104     // @formatter:off
105     String sql = ""
106         + "INSERT INTO orders ("
107         + "customer_fk, color_fk, engine_fk, tire_fk, model_fk, price"
108         + ") VALUES ("
109         + ":customer_fk, :color_fk, :engine_fk, :tire_fk, :model_fk, :price"
110         + ")";
111     // @formatter:on
112
113     SqlParams params = new SqlParams();
114
115     params.sql = sql;
116     params.source.addValue("customer_fk", customer.getCustomerPK());
117     params.source.addValue("color_fk", color.getColorPK());
118     params.source.addValue("engine_fk", engine.getEnginePK());
119     params.source.addValue("tire_fk", tire.getTirePK());
120     params.source.addValue("model_fk", jeep.getModelPK());
121     params.source.addValue("price", price);
122
123     return params;
124 }
125
126 /**
127 *
128 */
129 @Override
130 public List<Option> fetchOptions(List<String> optionIds) {
131     if (optionIds.isEmpty()) {
132         return new LinkedList<>();
133     }
134
135     Map<String, Object> params = new HashMap<>();
136
137     // @formatter:off
138     String sql = ""
139         + "SELECT * "
140         + "FROM options "
141         + "WHERE option_id IN(";
142     // @formatter:on
143 }
```

```
142     // @formatter:on
143
144     for (int index = 0; index < optionIds.size(); index++) {
145         String key = "option_" + index;
146         sql += ":" + key + ", ";
147         params.put(key, optionIds.get(index));
148     }
149
150     sql = sql.substring(0, sql.length() - 2);
151     sql += ")";
152
153     return jdbcTemplate.query(sql, params, new RowMapper<Option>() {
154         @Override
155         public Option mapRow(ResultSet rs, int rowNum) throws SQLException {
156             // @formatter:off
157             return Option.builder()
158                 .category(OptionType.valueOf(rs.getString("category")))
159                 .manufacturer(rs.getString("manufacturer"))
160                 .name(rs.getString("name"))
161                 .optionId(rs.getString("option_id"))
162                 .optionPK(rs.getLong("option_pk"))
163                 .price(rs.getBigDecimal("price"))
164                 .build();
165             // @formatter:on
166         }
167     });
168 }
169
170 /**
171 *
172 */
173
174 @Override
175 public Optional<Customer> fetchCustomer(String customerId) {
176     // @formatter:off
177     String sql = ""
178         + "SELECT * "
179         + "FROM customers "
180         + "WHERE customer_id = :customer_id";
181     // @formatter:on
182
183     Map<String, Object> params = new HashMap<>();
184     params.put("customer_id", customerId);
185
186     return Optional.ofNullable(
187         jdbcTemplate.query(sql, params, new CustomerResultSetExtractor()));
188 }
189
190 /**
191 */
192
193 @Override
194 public Optional<Jeep> fetchModel(JeepModel model, String trim, int doors) {
195     // @formatter:off
196     String sql = ""
197         + "SELECT * "
198         + "FROM models "
199         + "WHERE model_id = :model_id "
200         + "AND trim_level = :trim_level "
201         + "AND num_doors = :num_doors";
202     // @formatter:on
```

```
1 CreateOrderTest.java   2 JeepOrderController.java   3 *JeepOrderDao.java   4 DefaultJeepOrderDao.java X
201 // @formatter:on
202
203     Map<String, Object> params = new HashMap<>();
204     params.put("model_id", model.toString());
205     params.put("trim_level", trim);
206     params.put("num_doors", doors);
207
208     return Optional.ofNullable(
209         jdbcTemplate.query(sql, params, new ModelResultSetExtractor()));
210 }
211
212 /**
213 *
214 */
215 @Override
216 public Optional<Color> fetchColor(String colorId) {
217     // @formatter:off
218     String sql = ""
219         + "SELECT * "
220         + "FROM colors "
221         + "WHERE color_id = :color_id";
222     // @formatter:on
223
224     Map<String, Object> params = new HashMap<>();
225     params.put("color_id", colorId);
226
227     return Optional.ofNullable(
228         jdbcTemplate.query(sql, params, new ColorResultSetExtractor()));
229 }
230
231 /**
232 *
233 */
234 @Override
235 public Optional<Engine> fetchEngine(String engineId) {
236     // @formatter:off
237     String sql = ""
238         + "SELECT * "
239         + "FROM engines "
240         + "WHERE engine_id = :engine_id";
241     // @formatter:on
242
243     Map<String, Object> params = new HashMap<>();
244     params.put("engine_id", engineId);
245
246     return Optional.ofNullable(
247         jdbcTemplate.query(sql, params, new EngineResultSetExtractor()));
248 }
249
250 /**
251 *
252 */
253 @Override
254 public Optional<Tire> fetchTire(String tireId) {
255     // @formatter:off
256     String sql = ""
257         + "SELECT * "
258         + "FROM tires "
259         + "WHERE tire_id = :tire_id";
260     // @formatter:on
261 }
```

```
1 CreateOrderTest.java 2 JeepOrderController.java 3 *JeepOrderDao.java 4 DefaultJeepOrderDao.java X
261
262     Map<String, Object> params = new HashMap<>();
263     params.put("tire_id", tireId);
264
265     return Optional.ofNullable(
266         jdbcTemplate.query(sql, params, new TireResultSetExtractor()));
267 }
268
269 */
270 *
271 * @author Promineo
272 *
273 */
274
275 class TireResultSetExtractor implements ResultSetExtractor<Tire> {
276     @Override
277     public Tire extractData(ResultSet rs) throws SQLException {
278         rs.next();
279
280         // @formatter:off
281         return Tire.builder()
282             .manufacturer(rs.getString("manufacturer"))
283             .price(rs.getBigDecimal("price"))
284             .tireId(rs.getString("tire_id"))
285             .tirePK(rs.getLong("tire_pk"))
286             .tireSize(rs.getString("tire_size"))
287             .warrantyMiles(rs.getInt("warranty_miles"))
288             .build();
289         // @formatter:on
290     }
291 }
292
293 */
294 *
295 * @author Promineo
296 *
297 */
298 class EngineResultSetExtractor implements ResultSetExtractor<Engine> {
299     @Override
300     public Engine extractData(ResultSet rs) throws SQLException {
301         rs.next();
302
303         // @formatter:off
304         return Engine.builder()
305             .description(rs.getString("description"))
306             .engineId(rs.getString("engine_id"))
307             .enginePK(rs.getLong("engine_pk"))
308             .fuelType(FuelType.valueOf(rs.getString("fuel_type")))
309             .hasStartStop(rs.getBoolean("has_start_stop"))
310             .mpgCity(rs.getFloat("mpg_city"))
311             .mpgHwy(rs.getFloat("mpg_hwy"))
312             .name(rs.getString("name"))
313             .price(rs.getBigDecimal("price"))
314             .sizeInLiters(rs.getFloat("size_in_liters"))
315             .build();
316         // @formatter:on
317     }
318 }
```

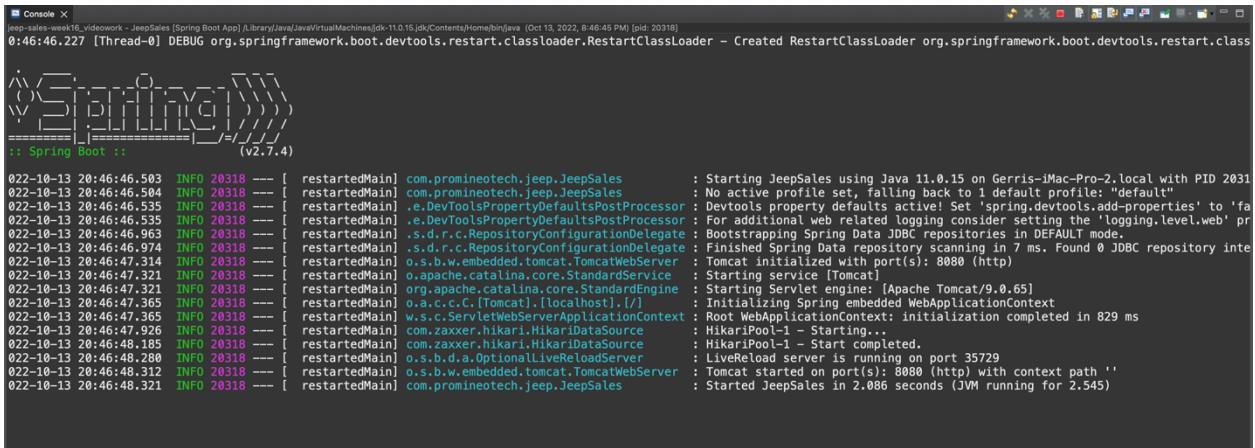
```
1 CreateOrderTest.java   2 JeepOrderController.java   3 *JeepOrderDao.java   4 DefaultJeepOrderDao.java X
318 /**
319  * @author Promineo
320  *
321  */
322 /**
323 */
324 class ColorResultSetExtractor implements ResultSetExtractor<Color> {
325     @Override
326     public Color extractData(ResultSet rs) throws SQLException {
327         rs.next();
328
329         // @formatter:off
330         return Color.builder()
331             .color(rs.getString("color"))
332             .colorId(rs.getString("color_id"))
333             .colorPK(rs.getLong("color_pk"))
334             .isExterior(rs.getBoolean("is_exterior"))
335             .price(rs.getBigDecimal("price"))
336             .build();
337         // @formatter:on
338     }
339 }
340 /**
341  *
342  * @author Promineo
343  *
344  */
345 /**
346 class ModelResultSetExtractor implements ResultSetExtractor<Jeep> {
347     @Override
348     public Jeep extractData(ResultSet rs) throws SQLException {
349         rs.next();
350
351         // @formatter:off
352         return Jeep.builder()
353             .basePrice(rs.getBigDecimal("base_price"))
354             .modelId(JeepModel.valueOf(rs.getString("model_id")))
355             .modelPK(rs.getLong("model_pk"))
356             .numDoors(rs.getInt("num_doors"))
357             .trimLevel(rs.getString("trim_level"))
358             .wheelSize(rs.getInt("wheel_size"))
359             .build();
360         // @formatter:on
361     }
362 }
363 /**
364  *
365  * @author Promineo
366  *
367  */
368 /**
369 class CustomerResultSetExtractor implements ResultSetExtractor<Customer> {
370     @Override
371     public Customer extractData(ResultSet rs) throws SQLException {
372         rs.next();
373     }

```

```

373
374     // @formatter:off
375     return Customer.builder()
376         .customerId(rs.getString("customer_id"))
377         .customerPK(rs.getLong("customer_pk"))
378         .firstName(rs.getString("first_name"))
379         .lastName(rs.getString("last_name"))
380         .phone(rs.getString("phone"))
381         .build();
382     // @formatter:on
383
384 }
385
386
387 class SqlParams {
388     String sql;
389     MapSqlParameterSource source = new MapSqlParameterSource();
390 }
391 }
```

Screenshots of Running Application:



The screenshot shows a terminal window with the title 'Console X'. The logs are from a Spring Boot application named 'JeepSales'. The output includes the application's startup banner, which features a stylized tree or mountain graphic. Below the banner, the log entries show the application starting up, including the configuration of Tomcat and the successful start of the application on port 8080.

```

0:46:46.227 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.class
:: Spring Boot ::
( v2.7.4 )
022-10-13 20:46:46.503 INFO 20318 --- [ restartedMain] com.promineotech.jeep.JeepSales      : Starting JeepSales using Java 11.0.15 on Gerris-iMac-Pro-2.local with PID 2031
022-10-13 20:46:46.504 INFO 20318 --- [ restartedMain] com.promineotech.jeep.JeepSales      : No active profile set, falling back to 1 default profile: "default"
022-10-13 20:46:46.535 INFO 20318 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : DevTools property defaults active! Set 'spring.devtools.add-properties' to 'false' for additional web related logging consider setting the 'logging.level.web' property
022-10-13 20:46:46.535 INFO 20318 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property
022-10-13 20:46:46.963 INFO 20318 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
022-10-13 20:46:46.974 INFO 20318 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 7 ms. Found 0 JDBC repositories
022-10-13 20:46:47.314 INFO 20318 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
022-10-13 20:46:47.321 INFO 20318 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
022-10-13 20:46:47.321 INFO 20318 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
022-10-13 20:46:47.365 INFO 20318 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
022-10-13 20:46:47.365 INFO 20318 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 829 ms
022-10-13 20:46:47.926 INFO 20318 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
022-10-13 20:46:48.185 INFO 20318 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
022-10-13 20:46:48.280 INFO 20318 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
022-10-13 20:46:48.312 INFO 20318 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
022-10-13 20:46:48.321 INFO 20318 --- [ restartedMain] com.promineotech.jeep.JeepSales      : Started JeepSales in 2.086 seconds (JVM running for 2.545)
```

The screenshot shows an IDE interface with two tabs: "JUnit X" and "Console X".

JUnit X:

- Runs: 1/1
- Errors: 0
- Failures: 0
- Finished after 4.654 seconds

Console X:

```

<terminated> FetchJeepTest (3) [JUnit] /Library/Java/JavaVirtualMachines/jdk-11.0.15.jdk/Contents/Home/bin/java (Oct 13, 2022, 8:46:07 PM - 8:46:12 PM) [pid: 20312]
20:46:07.843 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContext
20:46:07.850 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext
20:46:07.878 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContext
20:46:07.886 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Instantiating CacheAwareContext
20:46:07.889 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContext
20:46:07.890 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not find configuration class for context
20:46:07.890 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not find configuration class for context
20:46:07.890 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoader - Could not find configuration class for context
20:46:08.009 [main] DEBUG org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Found 1 candidate(s) for annotation [org.springframework.boot.test.context.SpringBootTest]
20:46:08.010 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Instantiating CacheAwareContext
20:46:08.087 [main] DEBUG org.springframework.boot.test.context.SpringBootTestBootstrapper - Instantiating TestContext
20:46:08.087 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Instantiating CacheAwareContext
20:46:08.087 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Instantiating TestContext
20:46:08.101 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper - Instantiating CacheAwareContext
20:46:08.103 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Initiating test execution
20:46:08.114 [main] DEBUG org.springframework.test.context.support.DependencyInjectionTestExecutionListener - Initiating test execution

.
```
(())
```
:: Spring Boot ::
(v2.7.4)

2022-10-13 20:46:08.418 INFO 20312 --- [           main] c.p.jeep.controller.FetchJeepTest      : 
2022-10-13 20:46:08.419 INFO 20312 --- [           main] c.p.jeep.controller.FetchJeepTest      : 
2022-10-13 20:46:08.937 INFO 20312 --- [           main] .s.d.r.c.RepositoryConfigurationDelegat : 
2022-10-13 20:46:08.951 INFO 20312 --- [           main] .s.d.r.c.RepositoryConfigurationDelegat : 
2022-10-13 20:46:09.422 INFO 20312 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : 
2022-10-13 20:46:09.434 INFO 20312 --- [           main] o.apache.catalina.core.StandardService : 
2022-10-13 20:46:09.434 INFO 20312 --- [           main] o.apache.catalina.core.StandardEngine : 
2022-10-13 20:46:09.535 INFO 20312 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : 
2022-10-13 20:46:09.535 INFO 20312 --- [           main] w.s.c.ServletWebServerApplicationContext : 
2022-10-13 20:46:10.911 INFO 20312 --- [           main] com.zaxxer.hikari.HikariDataSource : 
2022-10-13 20:46:11.148 INFO 20312 --- [           main] com.zaxxer.hikari.HikariDataSource : 
2022-10-13 20:46:11.374 INFO 20312 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : 
2022-10-13 20:46:11.386 INFO 20312 --- [           main] c.p.jeep.controller.FetchJeepTest      : 
2022-10-13 20:46:12.036 INFO 20312 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : 
2022-10-13 20:46:12.037 INFO 20312 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : 
2022-10-13 20:46:12.038 INFO 20312 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : 
2022-10-13 20:46:12.174 INFO 20312 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepSalesController : 
2022-10-13 20:46:12.181 INFO 20312 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepSalesService : 
2022-10-13 20:46:12.396 INFO 20312 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : 
2022-10-13 20:46:12.400 INFO 20312 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource :

```

URL to GitHub Repository:

<https://github.com/geraldinedepaul17/BackEndClass/tree/main/Week%2016>