

Web API Design with Spring Boot Week 16 Coding Assignment


Points possible: 75

URL to GitHub Repository: <https://github.com/f-guedes/jeep-sales.git>


URL to Public Link of your Video: https://youtu.be/sx-l_GXY1GE

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.

- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Use your existing repo or create a new repository on GitHub for this week's assignment and push your completed code to the repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the screenshots into this Assignment Document indicated by: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.


2. In addition, please include the following in your Coding Assignment Document:

- The requested screenshots, indicated by: 
- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
 - Upload the .pdf to the LMS in your Coding Assignment Submission.
-

Web API Design with Spring Boot Week 16 Coding Assignment

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 1b. For example:

Web API Design with Spring Boot Week 16 Coding Assignment

```
{
  "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. 

```
protected String createOrderBody() {
    // @formatter:off
    return "{\n"
        + "  \"customer\": \"IGNATOV_GISELLA\", \n"
        + "  \"model\": \"COMPASS\", \n"
        + "  \"trim\": \"80th Anniversary\", \n"
        + "  \"doors\": 4, \n"
        + "  \"color\": \"EXT_SPITFIRE_ORANGE\", \n"
        + "  \"engine\": \"2_0_TURBO\", \n"
        + "  \"tire\": \"235_CONTINENTAL\", \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
}
```

Web API Design with Spring Boot Week 16 Coding Assignment

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.jeeppromineotech.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");
```

Web API Design with Spring Boot Week 16 Coding Assignment

```
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);

    // And: the returned order is correct
    assertThat(response.getBody()).isNotNull();


    Order order = response.getBody();
    assertThat(order.getCustomer().getCustomerId()).isEqualTo("IGNATOV_GISELLA");
    assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.COMPASS);
    assertThat(order.getModel().getTrimLevel()).isEqualTo("80th Anniversary");
    assertThat(order.getModel().getNumDoors()).isEqualTo(4);
    assertThat(order.getColor().getColorId()).isEqualTo("EXT_SPITFIRE_ORANGE");
    assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
    assertThat(order.getTire().getTireId()).isEqualTo("235_CONTINENTAL");
    assertThat(order.getOptions()).hasSize(6);
}
```

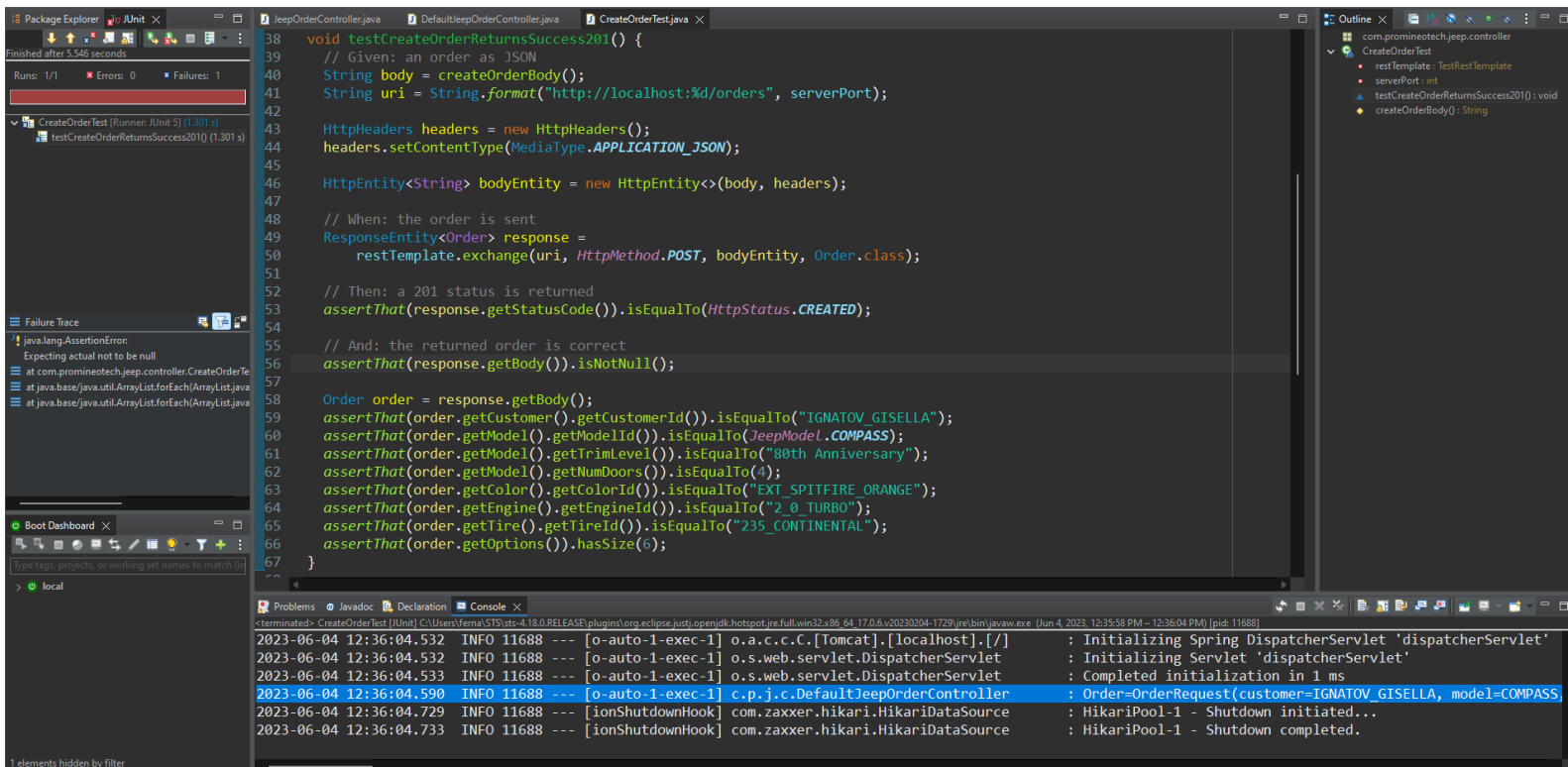
- 3) In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add @RequestMapping("/orders") as a class-level annotation.
- a) Create a method in the interface to create an order (createOrder). It should return an object of type Order (see below). It should accept a single parameter of type OrderRequest as described in the video. Make sure it accepts an HTTP POST request and returns a status code of 201 (created).
 - b) Add the @RequestBody annotation to the orderRequest parameter. Make sure to add the RequestBody annotation from the org.springframework.web.bind.annotation package.
 - c) Produce a screenshot of the finished JeepOrderController interface showing no compile errors. 

Web API Design with Spring Boot Week 16 Coding Assignment

```
1 package com.promineotech.jeepp.controller;
2
3 import org.springframework.http.HttpStatus;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 @RequestMapping("/orders")
20 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"),
21     servers = {@Server(url = "http://localhost:8080", description = "Jeep Order Service")},
22 public interface JeepOrderController {
23
24     // @formatter:off
25     @Operation(
26         summary = "Create an order for a Jeep.",
27         description = "Returns the created Jeep.",
28         responses = {
29             @ApiResponse(
30                 responseCode = "201",
31                 description = "The created Jeep is returned.",
32                 content = @Content(
33                     mediaType = "application/json",
34                     schema = @Schema(implementation = Order.class))
35             @ApiResponse(
36                 responseCode = "400",
37                 description = "The request parameters are invalid.",
38                 content = @Content(mediaType = "application/json"))
39             @ApiResponse(
40                 responseCode = "404",
41                 description = "A Jeep component was not found with",
42                 content = @Content(mediaType = "application/json"))
43             @ApiResponse(
44                 responseCode = "500",
45                 description = "An unplanned error occurred.",
46                 content = @Content(mediaType = "application/json"))
47         },
48         parameters = {
49             @Parameter(
50                 name = "orderRequest",
51                 required = true,
52                 description = "The order as JSON")
53         }
54     )
55     // @formatter:off
56
57     @PostMapping
58     @ResponseStatus(code = HttpStatus.CREATED)
59     Order createOrder(@RequestBody OrderRequest orderRequest);
60 }
```

Web API Design with Spring Boot Week 16 Coding Assignment

- 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. 



The screenshot shows an IDE with three tabs: `JeepOrderController.java`, `DefaultJeepOrderController.java`, and `CreateOrderTest.java`. The `CreateOrderTest.java` tab is active, showing a test method `testCreateOrderReturnsSuccess201()` with several assertions. The test is failing, as indicated by the red status bar in the JUnit runner on the left. The failure trace shows a `java.lang.AssertionError` with the message "Expecting actual not to be null". The console at the bottom shows the log output, including the log line `Order=OrderRequest(customer=IGNATOV_GISELLA, model=COMPASS)` and the error message `java.lang.AssertionError: Expecting actual not to be null`.

- 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]*")`

Web API Design with Spring Boot Week 16 Coding Assignment

b) Use these annotations for integer types:

i) `@Positive`

ii) `@Min(2)`


iii) `@Max(4)`

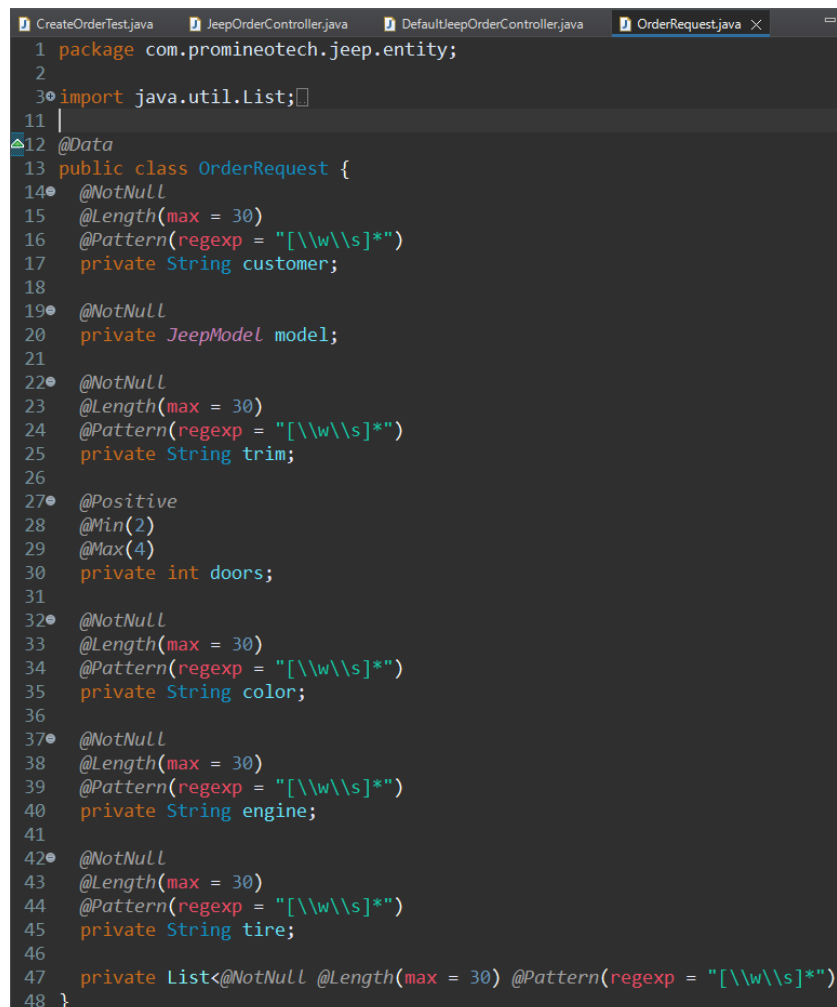
c) Add `@NotNull` to the enum type.

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. 



```
1 package com.promineotech.jee.entity;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12 @Data
13 public class OrderRequest {
14     @NotNull
15     @Length(max = 30)
16     @Pattern(regexp = "[\\w\\s]*")
17     private String customer;
18
19     @NotNull
20     private JeepModel model;
21
22     @NotNull
23     @Length(max = 30)
24     @Pattern(regexp = "[\\w\\s]*")
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 = "[\\w\\s]*")
35     private String color;
36
37     @NotNull
38     @Length(max = 30)
39     @Pattern(regexp = "[\\w\\s]*")
40     private String engine;
41
42     @NotNull
43     @Length(max = 30)
44     @Pattern(regexp = "[\\w\\s]*")
45     private String tire;
46
47     private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*")
48 }
```


Web API Design with Spring Boot Week 16 Coding Assignment

- 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).

```
1 package com.promineotech.jeep.service;
2
3 import org.springframework.stereotype.Service;
4 import com.promineotech.jeep.entity.Order;
5 import com.promineotech.jeep.entity.OrderRequest;
6 import lombok.extern.slf4j.Slf4j;
7
8 @Service
9 @Slf4j
10 public class DefaultJeepOrderService implements JeepOrderService {
11
12     @Override
13     public Order createOrder(OrderRequest orderRequest) {
14         log.info("Order={}", orderRequest);
15         return null;
16     }
17
18 }
```

```
2023-06-13 16:31:39.115 INFO 30188 --- [main] c.p.jeep.controller.CreateOrderTest : Starting CreateOrderTest using Java 17.0.6 on DESKTOP-00
2023-06-13 16:31:39.117 INFO 30188 --- [main] c.p.jeep.controller.CreateOrderTest : The following 1 profile is active: "test"
2023-06-13 16:31:40.842 INFO 30188 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 0 (http)
2023-06-13 16:31:40.853 INFO 30188 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-06-13 16:31:40.853 INFO 30188 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.74]
2023-06-13 16:31:40.989 INFO 30188 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-06-13 16:31:40.989 INFO 30188 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 16 ms
2023-06-13 16:31:42.312 INFO 30188 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 60622 (http) with context path=/
2023-06-13 16:31:42.327 INFO 30188 --- [main] c.p.jeep.controller.CreateOrderTest : Started CreateOrderTest in 3.663 seconds (JVM running for 10.171s)
2023-06-13 16:31:42.368 INFO 30188 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-06-13 16:31:42.675 INFO 30188 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-06-13 16:31:43.512 INFO 30188 --- [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-06-13 16:31:43.512 INFO 30188 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-06-13 16:31:43.514 INFO 30188 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
2023-06-13 16:31:43.677 INFO 30188 --- [o-auto-1-exec-1] c.p.j.service.DefaultJeepOrderService : Order=OrderRequest(customer=IGNATOV_GISELLA, model=COMPACT)
2023-06-13 16:31:43.809 INFO 30188 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2023-06-13 16:31:43.812 INFO 30188 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

Web API Design with Spring Boot Week 16 Coding Assignment


- 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:

Web API Design with Spring Boot Week 16 Coding Assignment

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. 

```
@Transactional
@Override
public Order createOrder(OrderRequest orderRequest) {
    Log.info("Order={}", orderRequest);

    Customer customer = getCustomer(orderRequest);
    Jeep jeep = getModel(orderRequest);
    Color color = getColor(orderRequest);
    Engine engine = getEngine(orderRequest);
    Tire tire = getTire(orderRequest);
    List<Option> options = getOption(orderRequest);

    BigDecimal price = jeep.getBasePrice().add(color.getPrice()).add(engine.getPrice()).add(tire.getPrice());

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

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

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

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

Web API Design with Spring Boot Week 16 Coding Assignment

```
@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
}
```

- 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. 🖥️

The screenshot displays an IDE interface with the following components:

- Package Explorer:** Shows the project structure with 'CreateOrderTest' selected under 'com.promineotech.jeepp.controller'.
- JUnit Status Bar:** Located at the bottom left, it shows a green bar indicating a successful test run. The text 'Runs: 1/1', 'Errors: 0', and 'Failures: 0' is visible.
- Test Class:** The 'CreateOrderTest.java' file is open in the editor. It contains a test method 'testCreateOrderReturnsSuccess201()' which uses 'restTemplate' to send a POST request and asserts the response status and body.
- Console:** The bottom panel shows the output of the test run, including timestamps, log levels (INFO), and messages from the application and the test framework.