


# 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.jeepp.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. 

```
3
4 class CreateOrderTest extends CreateOrderTestSupport {
5
6     @Test
7     void testCreateOrderReturnsSuccess201() {
8         // Given: an order as JSON
9         String body = createOrderBody();
10        String uri = getBaseUriForOrders();
11        HttpEntity<String> bodyEntity = new HttpEntity<>(body);
12
13        // When: the order is sent
14        ResponseEntity<?> response = getRestTemplate().exchange(uri, HttpMethod.POST,
15        // Then: a 201 status is returned
16        assertEquals(response.getStatusCode(), HttpStatus.CREATED);
17
18        // and: the returned order is correct
19    }
```

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.jeepp.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. 

```

@rest
void testCreateOrderReturnsSuccess201() {
    // Given: an order as JSON
    String body = createOrderBody();
    String uri = getBaseUriForOrders();

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

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

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


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

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

    Order order = response.getBody();
    assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");
    assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);
    assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
    assertThat(order.getModel().getNumDoors()).isEqualTo(4);
    assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");
    assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
    assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
    assertThat(order.getOptions()).hasSize(6);
}


```

- 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.

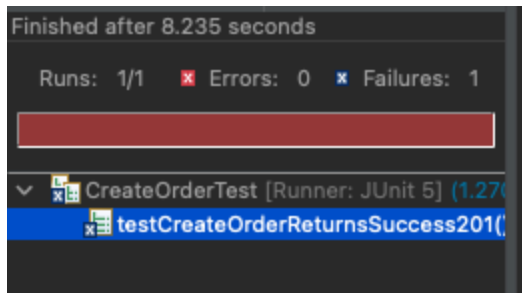
- c) Produce a screenshot of the finished JeepOrderController interface showing no compile errors. 

```
28
29 public interface JeepOrderController {
30
31
32     // @formatter:off
33     @Operation(
34         summary = "Create an order for a Jeep",
35         description = "Returns the created Jeep",
36         responses = {
37             @ApiResponse(
38                 responseCode = "201",
39                 description = "The created Jeep is returned",
40                 content = @Content(
41                     mediaType = "application/json",
42                     schema = @Schema(implementation = Order.class)
43             ),
44             @ApiResponse(
45                 responseCode = "400",
46                 description = "The request parameters are invalid",
47                 content = @Content(
48                     mediaType = "application/json")),
49             @ApiResponse(
50                 responseCode = "404",
51                 description = "A jeep component was not found with th
52                 content = @Content(
53                     mediaType = "application/json")),
54             @ApiResponse(
55                 responseCode = "500",
56                 description = "An unplanned error occurred",
57                 content = @Content(
58                     mediaType = "application/json"))
59         },
60         parameters = {
61             @Parameter(
62                 name = "orderRequest",
63                 required = true,
64                 description = "The order as JSON"),
65         }
66     )
67     @PostMapping
68     @ResponseStatus(code = HttpStatus.CREATED)
69     Order createORDER(@Valid @RequestBody OrderRequest request);
70     // @formatter:on
71 }
```

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

- Add @RestController as a class-level annotation.
- Add a log line to the implementing controller method showing the input request body (orderRequest)
- 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. 

**\*\*\*VIDEO ISN'T in same order as homework\*\*\***



```
class CreateOrderTest extends CreateOrderTestSupport {  
  
    @Test  
    void testCreateOrderReturnsSuccess201() {  
        // Given: an order as JSON  
        String body = createOrderBody();  
        String url = getBaseUrlForOrders();  
  
        HttpHeaders headers = new HttpHeaders();  
        headers.setContentType(MediaType.APPLICATION_JSON);  
  
        HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);  
  
        // When: the order is sent  
        ResponseEntity<Order> response = getRestTemplate().exchange(uri, HttpMethod.POST,  
        // Then: a 201 status is returned  
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);  
  
        // and: the returned order is correct  
        assertThat(response.getBody()).isNotNull();  
    }  
}
```

```
2022-06-09 23:04:53.247 INFO 3386 --- [main] c.p.jeepr.controller.CreateOrderTest  
2022-06-09 23:04:53.247 INFO 3386 --- [main] c.p.jeepr.controller.CreateOrderTest  
2022-06-09 23:04:58.678 INFO 3386 --- [main] c.p.jeepr.controller.CreateOrderTest  
2022-06-09 23:04:59.895 DEBUG 3386 --- [o-auto-1-exec-1] c.p.j.c.BasicJeepOrderController  
2022-06-09 23:04:59.926 ERROR 3386 --- [o-auto-1-exec-1] c.p.j.errorhandler.GlobalExceptionHandler  
  
org.springframework.web.util.NestedServletException: Handler dispatch failed; nested exception  
Type mismatch: cannot convert from Optional<Customer> to Customer
```

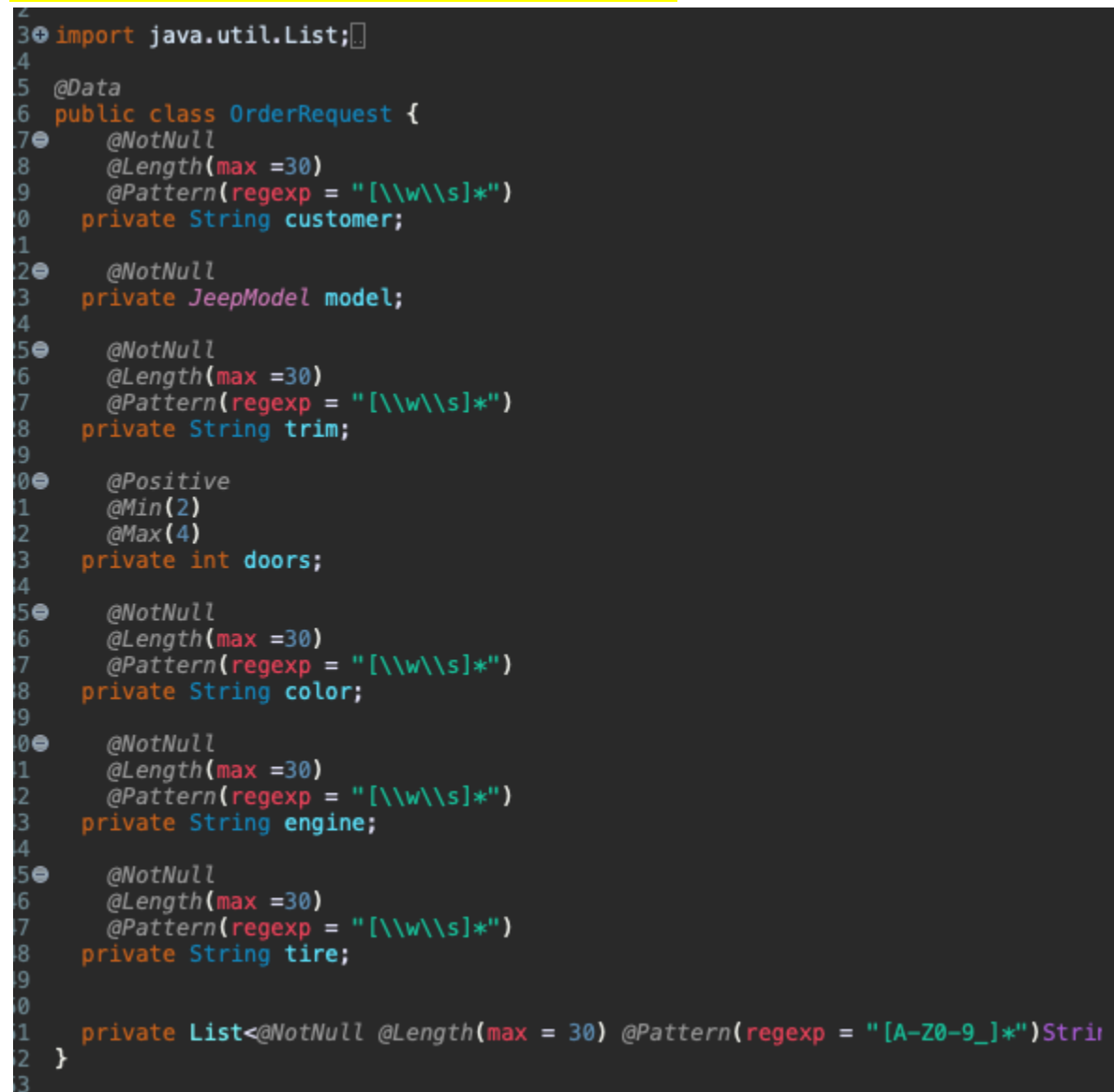
- 5) Find the Maven dependency spring-boot-starter-validation by looking it up at <https://mvnrepository.com/>. Add this repository to the project POM file (pom.xml).
- 6) Add the class-level annotation @Validated to the JeepOrderController interface.
- 7) Add Bean Validation annotations to the OrderRequest 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. 



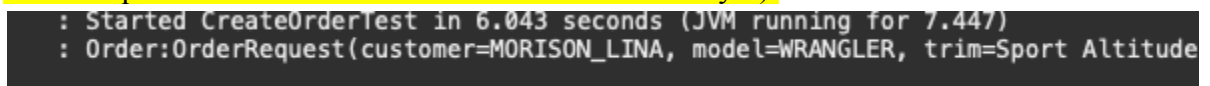
```
1 import java.util.List;
2
3 @Data
4 public class OrderRequest {
5     @NotNull
6     @Length(max = 30)
7     @Pattern(regexp = "[\\w\\s]*")
8     private String customer;
9
10    @NotNull
11    private JeepModel model;
12
13    @NotNull
14    @Length(max = 30)
15    @Pattern(regexp = "[\\w\\s]*")
16    private String trim;
17
18    @Positive
19    @Min(2)
20    @Max(4)
21    private int doors;
22
23    @NotNull
24    @Length(max = 30)
25    @Pattern(regexp = "[\\w\\s]*")
26    private String color;
27
28    @NotNull
29    @Length(max = 30)
30    @Pattern(regexp = "[\\w\\s]*")
31    private String engine;
32
33    @NotNull
34    @Length(max = 30)
35    @Pattern(regexp = "[\\w\\s]*")
36    private String tire;
37
38    private List<@NotNull @Length(max = 30) @Pattern(regexp = "[A-Z0-9_]*") String>
39    options;
40 }
```

- 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).



```
: Started CreateOrderTest in 6.043 seconds (JVM running for 7.447)
: Order:OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport Altitude)
```

- 9) In the `jeep.dao` sub-package, create the empty (no methods yet) DAO interface (named `JeepOrderDao`) and implementation (named `DefaultJeepOrderDao`).
  - a) Inject the DAO interface into the order service implementation class.
  - b) Add the `@Component` annotation to the DAO implementation class.
- 10) Replace the entire content of `JeepOrderDao.java` with the source found in `JeepOrderDao.source`. The source file is found in the Source folder of the supplied project resources.

**11) \*\*\* The next steps require you to copy source code from the Source directory in the supplied resources. Please follow the instructions EXACTLY. Some steps require you to replace ALL the source in a file. Some steps require you to ADD source to a file.**

- 12) Copy the *contents* of the file `DefaultJeepOrderDao.source` into `DefaultJeepOrderDao.java`. The source file is found in the Source folder of the supplied project resources.

In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable. Make sure you pick the import `java.util.Optional`, `java.util.List`, and `org.springframework.jdbc.core.RowMapper`.

- 13) Copy the *contents* of the file `DefaultJeepOrderService.source` into `DefaultJeepOrderService.java`. Add the source after the `createOrder()` method, but *inside* the class body. The source file is found in the Source folder of the supplied project resources.

In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable.

- 14) In `DefaultJeepOrderService.java`, work with the method `createOrder`.
  - a) Add the `@Transactional` annotation to the `createOrder` method.



- b) In the createOrder method call the copied methods: getCustomer, getModel, getColor, getEngine, getTire and getOption, assigning the return values of these methods to variables of the appropriate types.
- c) Calculate the price, including all options.

15) In JeepOrderDao.java and DefaultJeepOrderDao.java, add the method:

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

- a) Call the jeepOrder.Dao.saveOrder method from the jeepOrderSalesService.createOrder service. Produce a screenshot of the jeepOrderSalesService.createOrder method.

```
package com.promineotech.jeepp.service;  
  
import com.promineotech.jeepp.entity.Order;  
  
public interface JeepOrderService {  
    Order createOrder(OrderRequest orderRequest);  
}
```

```
1 package com.promineotech.jeepp.service;  
2  
3 import java.math.BigDecimal;  
20  
21 @Service  
22 public class DefaultJeepOrderService implements JeepOrderService {  
23  
24     @Autowired  
25     private JeepOrderDao jeepOrderDao;  
26  
27     @Transactional  
28     @Override  
29     public Order createOrder(OrderRequest orderRequest) {  
30
```

```

@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.getBasePrice());

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

    return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price,
}

```

```

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

protected Tire getTire(OrderRequest orderRequest) {
    return jeepOrderDao.fetchTire(orderRequest.getTire()).orElseThrow(
        () -> new NoSuchElementException("Tire with ID= " + orderRequest.getTireId())
    );
}

protected Engine getEngine(OrderRequest orderRequest) {
    return jeepOrderDao.fetchEngine(orderRequest.getEngine()).orElseThrow(
        () -> new NoSuchElementException("Engine with ID= " + orderRequest.getEngineId())
    );
}

protected Color getColor(OrderRequest orderRequest) {
    return jeepOrderDao.fetchColor(orderRequest.getColor()).orElseThrow(
        () -> new NoSuchElementException("Color with ID=" + orderRequest.getColorId())
    );
}

protected Jeep getModel(OrderRequest orderRequest) {
    return jeepOrderDao.fetchModel(orderRequest.getModel(), orderRequest.getTrim(), orderRequest.getDoors(),
        .orElseThrow(() -> new NoSuchElementException("Model with Id= " + orderRequest.getModelId()
            + orderRequest.getTrim() + orderRequest.getDoors() + " was not found"))
    );
}

protected Customer getCustomer(OrderRequest orderRequest) {
    return jeepOrderDao.fetchCustomer(orderRequest.getCustomer()).orElseThrow(
        () -> new NoSuchElementException("Customer with ID= " + orderRequest.getCustomerId())
    );
}

```

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. 

```
@Override
public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine,
    Tire tire, Price price) {
    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
}

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

- 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.jee.controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
6 @ActiveProfiles("test")
7 @Sql(scripts = { "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
8                 "classpath:flyway/migrations/V1.1__Jeep_Data.sql" }, config = @SqlConfig(encrypt = false))
9 class CreateOrderTest extends CreateOrderTestSupport {
10
11     @Test
12     void testCreateOrderReturnsSuccess201() {
13         // Given: an order as JSON
14         String body = createOrderBody();
15         String uri = getBaseUrlForOrders();
16
17         HttpHeaders headers = new HttpHeaders();
18         headers.setContentType(MediaType.APPLICATION_JSON);
19
20         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
21
22         // When: the order is sent
23         ResponseEntity<Order> response = getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity, Order.class);
24
25         // Then: a 201 status is returned
26         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
27
28         // and: the returned order is correct
29         assertThat(response.getBody()).isNotNull();
30     }
31 }
```

Finished after 8.332 seconds

Runs: 1/1 Errors: 0 Failures: 0

Build SUCCESSFUL

Spring Boot (v2.6.7)

2022-06-10 10:11:11.322 INFO 1801 --- [main] c.p.jee.controller.CreateOrderTest: testCreateOrderReturnsSuccess201() started

2022-06-10 10:11:11.323 DEBUG 1801 --- [main] c.p.jee.controller.CreateOrderTest: testCreateOrderReturnsSuccess201() body: {\"id\":1,\"make\":\"Jeep\",\"model\":\"Wrangler\",\"year\":2022,\"price\":45000}

2022-06-10 10:11:11.324 INFO 1801 --- [main] c.p.jee.controller.CreateOrderTest: testCreateOrderReturnsSuccess201() response: {\"id\":1,\"make\":\"Jeep\",\"model\":\"Wrangler\",\"year\":2022,\"price\":45000}

2022-06-10 10:11:16.927 INFO 1801 --- [main] c.p.jee.controller.CreateOrderTest: testCreateOrderReturnsSuccess201() finished

2022-06-10 10:11:18.103 DEBUG 1801 --- [o-auto-1-exec-1] c.p.j.c.BasicJeepOrderController: testCreateOrderReturnsSuccess201() finished

Screenshots of Code: see above

Screenshots of Running Application: see above

URL to GitHub Repository: <https://github.com/cperrine19/JeepSalesProject>