


# 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.

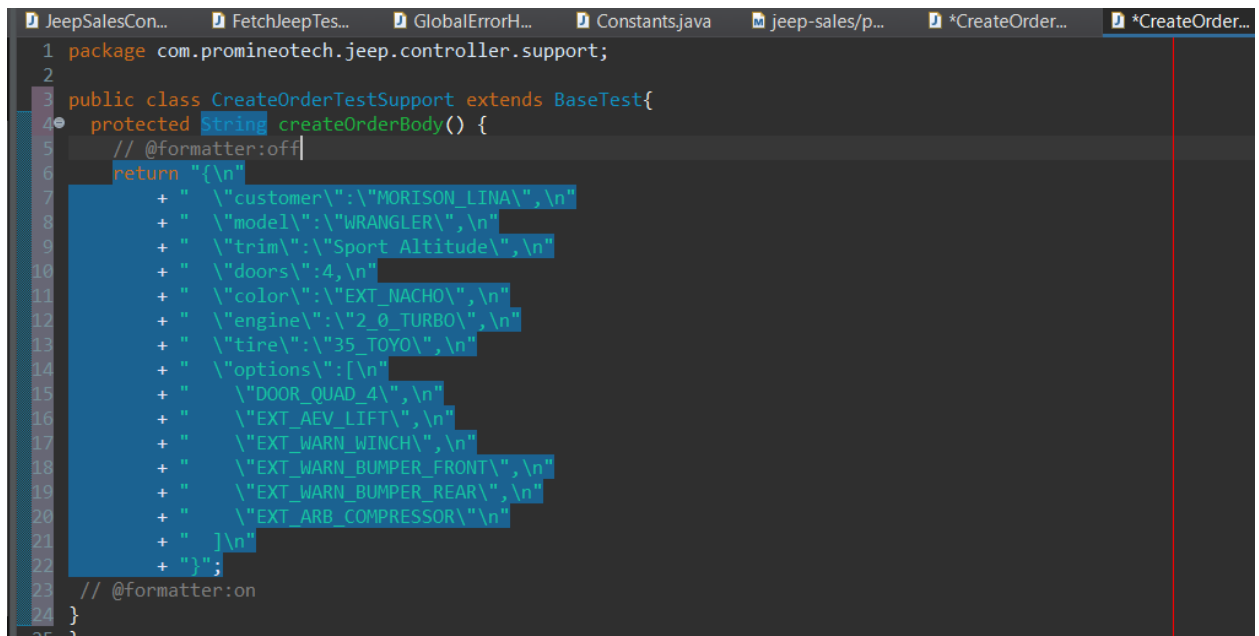
1) Select some options for a Jeep order:

- 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.jeeep.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. 



```
1 package com.promineotech.jeeep.controller.support;
2
3 public class CreateOrderTestSupport extends BaseTest{
4     protected String createOrderBody() {
5         // @formatter:off
6         return "{\n"
7             + "  \"customer\": \"MORISON_LINA\", \n"
8             + "  \"model\": \"WRANGLER\", \n"
9             + "  \"trim\": \"Sport Altitude\", \n"
10            + "  \"doors\": 4, \n"
11            + "  \"color\": \"EXT_NACHO\", \n"
12            + "  \"engine\": \"2_0_TURBO\", \n"
13            + "  \"tire\": \"35_TOYO\", \n"
14            + "  \"options\": [\n"
15            + "    \"DOOR_QUAD_4\", \n"
16            + "    \"EXT_AEV_LIFT\", \n"
17            + "    \"EXT_WARN_WINCH\", \n"
18            + "    \"EXT_WARN BUMPER_FRONT\", \n"
19            + "    \"EXT_WARN BUMPER_REAR\", \n"
20            + "    \"EXT_ARB_COMPRESSOR\" \n"
21            + "  ] \n"
22            + "}";
23         // @formatter:on
24     }
25 }
```

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.jeeep.entity.Order and not some other Order class.

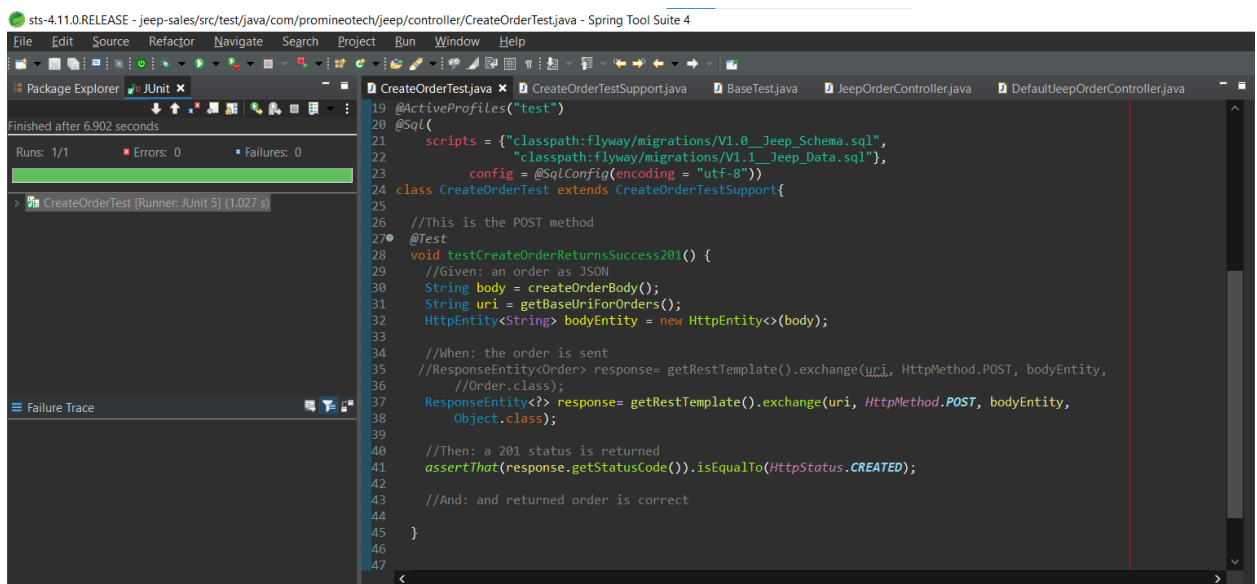
```
ResponseEntity<Order> response = restTemplate.exchange(uri,
    HttpMethod.POST, bodyEntity, Order.class);
```

- 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. 



```
sts-4.11.0.RELEASE - jeep-sales/src/test/java/com/promineotech/jeep/controller/CreateOrderTest.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit
Finished after 6.902 seconds
Runs: 1/1 Errors: 0 Failures: 0
CreateOrderTest [Runner: JUnit 5] (1.027 s)
Failure Trace
CreateOrderTest.java
19 @ActiveProfiles("test")
20 @Sql(
21     scripts = {"classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
22               "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
23     config = @SqlConfig(encoding = "utf-8"))
24 class CreateOrderTest extends CreateOrderTestSupport{
25
26     //This is the POST method
27     @Test
28     void testCreateOrderReturnsSuccess201() {
29         //Given: an order as JSON
30         String body = createOrderBody();
31         String uri = getBaseUrlForOrders();
32         HttpEntity<String> bodyEntity = new HttpEntity<>(body);
33
34         //When: the order is sent
35         //ResponseEntity<Order> response= getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity,
36         //Order.class);
37         ResponseEntity<> response= getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity,
38         Object.class);
39
40         //Then: a 201 status is returned
41         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
42
43         //And: and returned order is correct
44     }
45
46
47
```

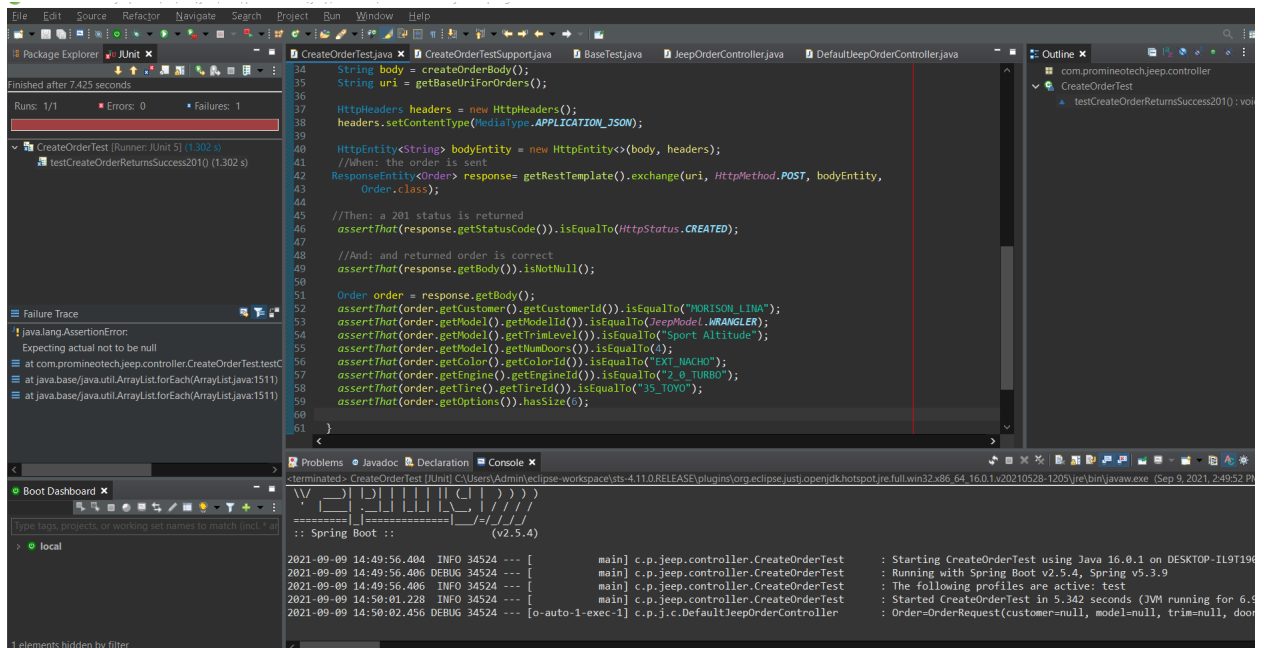
- 3) In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add @RequestMapping("/orders") as a class-level annotation.
- a) Create a method in the interface to create an order (createOrder). It should return an object of type Order (see below). It should accept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

```
CreateOrderTest.java  CreateOrderTestSupport.java  BaseTest.java  JeepOrderController.java  DefaultJeepOrderController.java
1 package com.promineotech.jeepp.controller;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.validation.annotation.Validated;
5 import org.springframework.web.bind.annotation.PostMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8 import com.promineotech.jeepp.entity.Order;
9 import com.promineotech.jeepp.entity.OrderRequest;
10 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
11 import io.swagger.v3.oas.annotations.Operation;
12 import io.swagger.v3.oas.annotations.Parameter;
13 import io.swagger.v3.oas.annotations.info.Info;
14 import io.swagger.v3.oas.annotations.media.Content;
15 import io.swagger.v3.oas.annotations.media.Schema;
16 import io.swagger.v3.oas.annotations.parameters.RequestBody;
17 import io.swagger.v3.oas.annotations.responses.ApiResponse;
18 import io.swagger.v3.oas.annotations.servers.Server;
19
20 @Validated
21 @RequestMapping("/orders")
22 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
23     @Server(url = "http://localhost:8080", description = "Local server.")})
24 public interface JeepOrderController {
25
26
27     // @formatter:off
28     @Operation(
29         summary = "Create an Order for a Jeep",
30         description = "Returns the created Jeep",
31         responses = {
32             @ApiResponse(
33                 responseCode = "201",
34                 description = "A created Jeep is returned",
35                 content = @Content(
36                     mediaType = "application/json",
37                     schema = @Schema(implementation = Order.class)),
38             @ApiResponse(
39                 responseCode = "400",
40                 description = "The request parameters are invalid",
41                 content = @Content(mediaType = "application/json"))
42         }
43     )
44 }
```

```
23 @Server(url = "http://localhost:8080", description = "Local server.")
24 public interface JeepOrderController {
25
26
27 // @formatter:off
28 @Operation(
29     summary = "Create an Order for a Jeep",
30     description = "Returns the created Jeep",
31     responses = {
32         @ApiResponse(
33             responseCode = "201",
34             description = "A created Jeep is returned",
35             content = @Content(
36                 mediaType = "application/json",
37                 schema = @Schema(implementation = Order.class)),
38         @ApiResponse(
39             responseCode = "400",
40             description = "The request parameters are invalid",
41             content = @Content(mediaType = "application/json")),
42         @ApiResponse(
43             responseCode = "404",
44             description = "A Jeeps component was not found with the input criteria",
45             content = @Content(mediaType = "application/json")),
46         @ApiResponse(
47             responseCode = "500",
48             description = "An unplanned error occurred.",
49             content = @Content(mediaType = "application/json"))
50     },
51     parameters = {
52         @Parameter(name = "orderRequest",
53             required = true,
54             description = "The order as JSON")
55     }
56 )
57 @PostMapping
58 @ResponseStatus(code = HttpStatus.CREATED)
59 Order createOrder(@RequestBody OrderRequest orderRequest);
60 // @formatter:on
61 }
62
63
```

- 4) Create a class that implements JeepOrderController named DefaultJeepOrderController.
  - a) Add @RestController as a class-level annotation.
  - b) Add a log line to the implementing controller method showing the input request body (orderRequest)

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



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

- a) Use these annotations for String types:

- i) `@NotNull`
- ii) `@Length(max = 30)`
- iii) `@Pattern(regexp = "[\\w\\s]*")`

- b) Use these annotations for integer types:

- i) `@Positive`
- ii) `@Min(2)`
- iii) `@Max(4)`


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

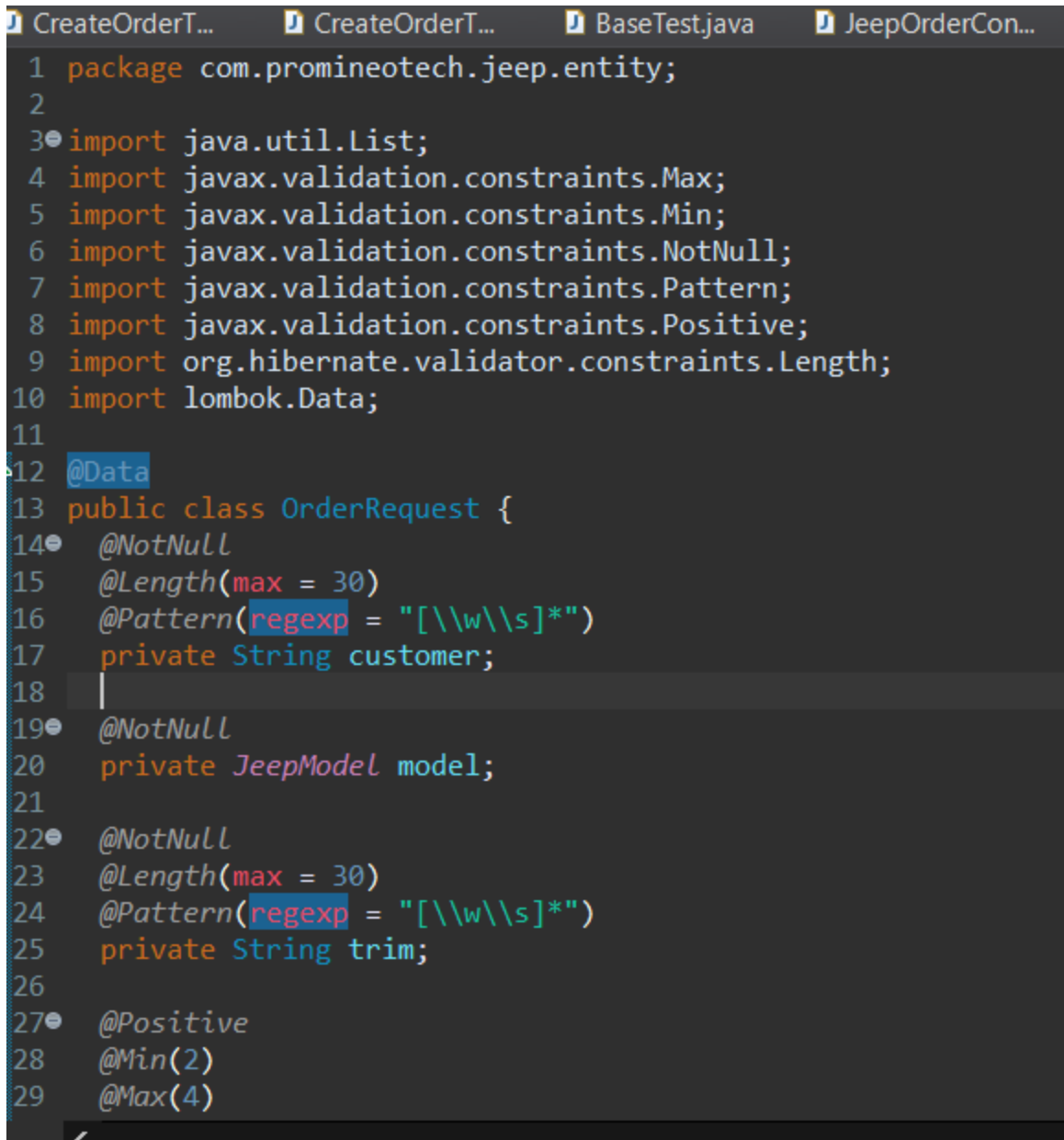
- d) Add validation to the list element (type String) by adding the validation annotations *inside* the generic definition. So, to add the String validation to the options, you would do this:

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

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



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



```
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(regex = "[\\w\\s]*")
17     private String customer;
18
19     @NotNull
20     private JeepModel model;
21
22     @NotNull
23     @Length(max = 30)
24     @Pattern(regex = "[\\w\\s]*")
25     private String trim;
26
27     @Positive
28     @Min(2)
29     @Max(4)
```

```
CreateOrderTest.j...  DeafultJeepOrde...  OrderRequest.java x  DefaultJee

23  @Length(max = 30)
24  @Pattern(regex = "[\\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(regex = "[\\w\\s]*")
35  private String color;
36
37  @NotNull
38  @Length(max = 30)
39  @Pattern(regex = "[\\w\\s]*")
40  private String engine;
41
42  @NotNull
43  @Length(max = 30)
44  @Pattern(regex = "[\\w\\s]*")
45  private String tire;
46
47
48  private List<@Length(max = 30)@Pattern(
49      regex = "[\\w\\s]*")String> options;
50  }
51
```

- 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);
```



```

18:59:37.618 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper - @TestExecutionListeners is not present for class [com.promineotech.jee
18:59:37.619 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Loaded default TestExecutionListener class names from location [META-INF/spring.factories
18:59:37.639 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.web.ServletTestExecution
18:59:37.643 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTestContext@698122b2 testClass = Create
18:59:37.661 [main] DEBUG org.springframework.test.context.support.DependencyInjectionTestExecutionListener - Performing dependency injection for test context [[DefaultTestContext@698122b
18:59:37.695 [main] DEBUG org.springframework.core.env.StandardEnvironment - Activating profiles [test]
18:59:37.702 [main] DEBUG org.springframework.test.context.support.TestPropertySourceUtils - Adding inlined properties to environment: {spring.jmx.enabled=false, org.springframework.boot.

:: Spring Boot ::
(v2.5.4)

2021-09-09 18:59:38.259 INFO 49376 --- [main] c.p.jee.controller.CreateOrderTest : Starting CreateOrderTest using Java 16.0.1 on DESKTOP-IL9T190 with PID 49376 (started
2021-09-09 18:59:38.260 DEBUG 49376 --- [main] c.p.jee.controller.CreateOrderTest : Running with Spring Boot v2.5.4, Spring v5.3.9
2021-09-09 18:59:38.260 INFO 49376 --- [main] c.p.jee.controller.CreateOrderTest : The following profiles are active: test
2021-09-09 18:59:43.678 INFO 49376 --- [main] c.p.jee.controller.CreateOrderTest : Started CreateOrderTest in 5.971 seconds (JVM running for 7.573)
2021-09-09 18:59:45.021 DEBUG 49376 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport Altitude, doors=4
2021-09-09 18:59:45.024 DEBUG 49376 --- [o-auto-1-exec-1] c.p.jee.dao.DefaultJeepOrderDao : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport Altitude, doors=4

```

```

package com.promineotech.jee.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.promineotech.jee.dao.JeeOrderDao;
import com.promineotech.jee.entity.Order;
import com.promineotech.jee.entity.OrderRequest;

@Service
public class DefaultJeepOrderService implements JeeOrderService {
    @Autowired
    private JeeOrderDao jeepOrderDao;

    @Override
    public Order createOrder(OrderRequest orderRequest) {
        return jeepOrderDao.createOrder(orderRequest);
    }
}

```

- 9) In the jeep.dao sub-package, create the empty (no methods yet) DAO interface (named JeepOrderDao) and implementation (named DefaultJeepOrderDao).
  - a) Inject the DAO interface into the order service implementation class.
  - b) Add the @Component annotation to the DAO implementation class.
- 10) Replace the entire content of JeepOrderDao.java with the source found in JeepOrderDao.source. The source file is found in the Source folder of the supplied project resources.
- 11) \*\*\* The next steps require you to copy source code from the Source directory in the supplied resources. Please follow the instructions EXACTLY. Some steps require you to replace ALL the source in a file. Some steps require you to ADD source to a file.

- 12) Replace the entire contents of `DefaultJeepOrderDao.java` with the source found in `DefaultJeepOrderDao.source`. The source file is found in the Source folder of the supplied project resources. After this step you will see errors in `DefaultJeepOrderDao`. This will be fixed shortly.
- 13) Copy the *contents* of the file `DefaultJeepOrderDao.source` *into* `DefaultJeepOrderDao.java`. The source file is found in the Source folder of the supplied project resources.

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

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

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

- 15) In `DefaultJeepOrderService.java`, work with the method `createOrder`.
- a) Add the `@Transactional` annotation to the `createOrder` method.
  - b) In the `createOrder` method call the copied methods: `getCustomer`, `getModel`, `getColor`, `getEngine`, `getTire` and `getOption`, assigning the return values of these methods to variables of the appropriate types.
  - c) Calculate the price, including all options.

- 16) In `JeepOrderDao.java` and `DefaultJeepOrderDao.java`, add the method:

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

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

The screenshot displays the Spring Tool Suite 4 IDE with the `DefaultJeepOrderService.java` file open. The code defines a `createOrder` method that takes an `OrderRequest` and returns an `Order`. It uses various getters to retrieve customer, jeep, color, engine, and tire information, and calculates the total price by summing the base price and individual component prices.

```
public Order createOrder(OrderRequest orderRequest) {
    List<Option> options = getOption(orderRequest);
    Customer customer = getCustomer(orderRequest);
    Jeep jeep = getJeep(orderRequest);
    Color color = getColor(orderRequest);
    Engine engine = getEngine(orderRequest);
    Tire tire = getTire(orderRequest);

    //To get price
    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);
}
```

The console output shows the execution of the `createOrder` method, including the start of the application, the running of the test, and the final output of the order details.

```
2021-09-13 13:55:22.668 INFO 60056 --- [main] c.p.jee... : Starting CreateOrderTest using Java 16.0.1 on DESKTOP-IL9T190 with P...
2021-09-13 13:55:22.670 INFO 60056 --- [main] c.p.jee... : Running with Spring Boot v2.5.4, Spring v5.3.9
2021-09-13 13:55:22.670 INFO 60056 --- [main] c.p.jee... : The following profiles are active: test
2021-09-13 13:55:27.545 INFO 60056 --- [main] c.p.jee... : Started CreateOrderTest in 5.458 seconds (JVM running for 7.159)
2021-09-13 13:55:28.666 DEBUG 60056 --- [o-auto-1-exec-1] c.p.j.c.DefaultJeepOrderController : Order=OrderRequest(customer=MORISON_LINA, model=WRANGLER, trim=Sport
```

```
CreateOrderTest.java  DefaultJeepOrderService.java x  JeepOrderService.java
1 package com.promineotech.jeep.service;
2
3 import java.math.BigDecimal;
4 import java.util.List;
5 import java.util.NoSuchElementException;
6 import java.util.Optional;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Service;
9 import org.springframework.transaction.annotation.Transactional;
10 import com.promineotech.jeep.dao.JeepOrderDao;
11 import com.promineotech.jeep.entity.Color;
12 import com.promineotech.jeep.entity.Customer;
13 import com.promineotech.jeep.entity.Engine;
14 import com.promineotech.jeep.entity.Jeep;
15 import com.promineotech.jeep.entity.Option;
16 import com.promineotech.jeep.entity.Order;
17 import com.promineotech.jeep.entity.OrderRequest;
18 import com.promineotech.jeep.entity.Tire;
19
20 @Service
21 public class DefaultJeepOrderService implements JeepOrderService {
22     @Autowired
23     private JeepOrderDao jeepOrderDao;
24
25     @Override
26     @Transactional
27     public Order createOrder(OrderRequest orderRequest) {
28         List<Option> options = getOption(orderRequest);
29         Customer customer = getCustomer(orderRequest);
30         Jeep jeep = getModel(orderRequest);
31         Color color = getColor(orderRequest);
32         Engine engine = getEngine(orderRequest);
33         Tire tire = getTire(orderRequest);
34
35         //To get price
36         BigDecimal price = jeep.getBasePrice()
37             .add(color.getPrice())
38             .add(engine.getPrice())
39             .add(tire.getPrice());
```

```
CreateOrderTest.java  DefaultJeepOrderService.java x  JeepOrderService.java
43     }
44
45     return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
46
47 }
48
49
50 /**
51  *
52  * @param orderRequest
53  * @return
54  */
55 private List<Option> getOption(OrderRequest orderRequest) {
56     return jeepOrderDao.fetchOptions(orderRequest.getOptions());
57 }
58
59 /**
60  *
61  * @param orderRequest
62  * @return
63  */
64 private Tire getTire(OrderRequest orderRequest) {
65     return jeepOrderDao.fetchTire(orderRequest.getTire())
66         .orElseThrow(() -> new NoSuchElementException(
67             "Tire with ID=" + orderRequest.getTire() + " was not found"));
68 }
69
70 /**
71  *
72  * @param orderRequest
73  * @return
74  */
75 private Engine getEngine(OrderRequest orderRequest) {
76     return jeepOrderDao.fetchEngine(orderRequest.getEngine())
77         .orElseThrow(() -> new NoSuchElementException(
78             "Engine with ID=" + orderRequest.getEngine() + " was not found"));
79 }
80
81 /**
```



```

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

```

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.

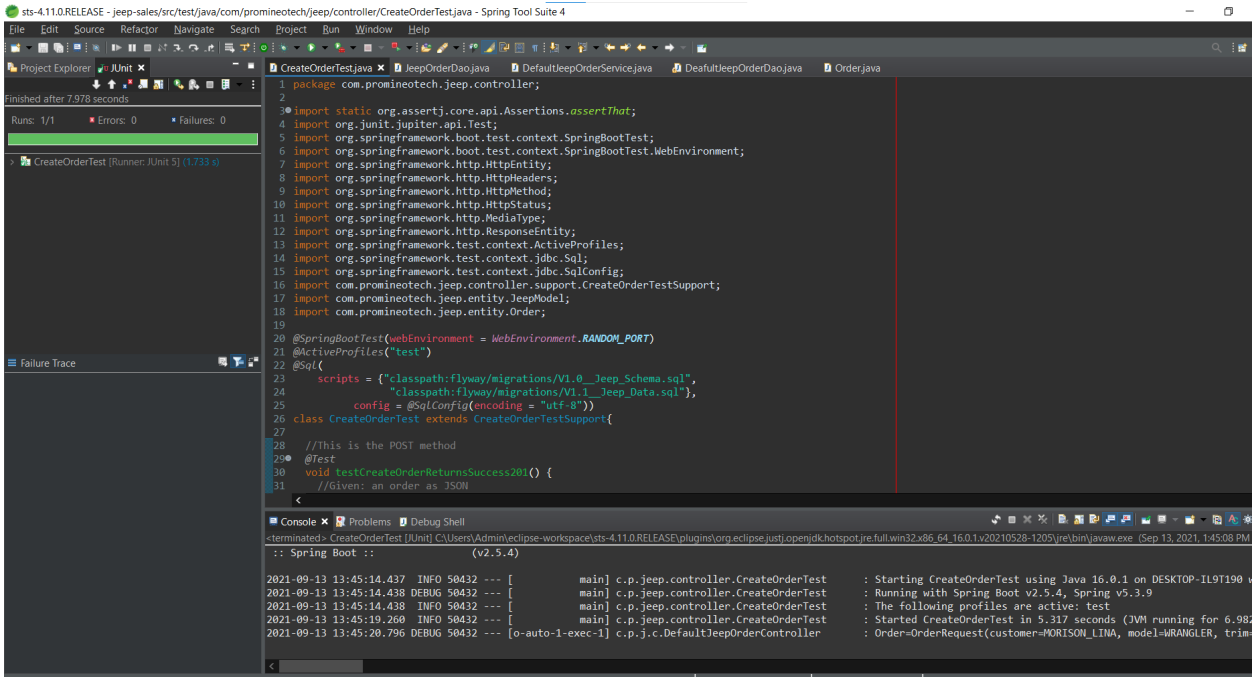
- iii) Write a method named saveOptions as shown in the video. This method should have the following method signature:

```
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. 

```
41
42● @Override
43 public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
44     BigDecimal price, List<Option> options) {
45     SqlParams params =
46         generateInsertSql(customer, jeep, color, engine, tire, price);
47
48     KeyHolder keyHolder = new GeneratedKeyHolder();
49     jdbcTemplate.update(params.sql, params.source, keyHolder);
50
51     Long orderPK = keyHolder.getKey().longValue();
52     saveOptions(options, orderPK);
53     // @formatter:off
54     return Order.builder()
55         .orderPK(orderPK)
56         .customer(customer)
57         .model(jeep)
58         .color(color)
59         .engine(engine)
60         .tire(tire)
61         .options(options)
62         .price(price)
63         .build();
64     // @formatter:on
65 }
66● /**
67  *
```

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

Screenshots of Code:

Screenshots of Running Application:

URL to GitHub Repository:

<https://github.com/JoleneMel/Jeep-SalesW4>