

**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.

- Select some options for a Jeep order:
  - 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:
    - color
    - customer
    - engine
    - model
    - tire(s)
  - 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!
- 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.
  - 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":"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(



```

protected String createOrderBody() {
    // @formatter:off
    return "{\n"
        + "  \"customer\": \"ATTAWAY_HECKTOR\", \n"
        + "  \"model\": \"WRANGLER\", \n"
        + "  \"trim\": \"Sport Altitude\", \n"
        + "  \"doors\": 4, \n"
        + "  \"color\": \"EXT_DIAMOND_BLACK\", \n"
        + "  \"engine\": \"2_0_HYBRID\", \n"
        + "  \"tire\": \"265_MICHELIN\", \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
}

```

) method.

In the test method, assign the return value of the createOrderBody() method to a variable named body.

- 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.
- Add another instance variable for an injected TestRestTemplate named restTemplate.
- In the test method, assign a value to a local variable named uri as follows:

```
String uri = String.format("http://localhost:%d/orders", serverPort);
```

- 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`.

- Create an `HttpEntity` object and set the request body and headers:

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

- 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);
```

- 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);
```

- Produce a screenshot of the tes

```
@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);
    // Given: an order as JSON
    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("ATTAWAY_HECKTOR");
    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_DIAMOND_BLACK");
    assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_HYBRID");
    assertThat(order.getTire().getTireId()).isEqualTo("265_MICHELIN");
    assertThat(order.getOptions()).hasSize(6);
}
```

t method.

- In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add `@RequestMapping("/orders")` as a class-level annotation.
  - 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).
  - Add the `@RequestBody` annotation to the orderRequest parameter. Make sure to add the RequestBody annotation from the org.springframework.web.bind.annotation package.
  - Produce a screenshot of the finished JeepOrderController interface showing no compile errors.

```

1 package com.promineotech.jeep.controller;
2
3 import javax.validation.Valid;
4
5 @Validated
6 @RequestMapping("/orders")
7 @OpenAPIDefinition(info = @Info(title = "Jeep Order Service", servers = {
8     @Server(url = "http://localhost:8080", description = "Local server.")})
9
10 public interface JeepOrderController {
11
12     // @formatter: off
13     @Operation(
14         summary = "Create an order for a Jeep",
15         description = "Returns the created Jeep.",
16         responses = {
17             @ApiResponse(responseCode = "201",
18                 description = "The created Jeep is returned",
19                 content = @Content(
20                     mediaType = "application/json",
21                     schema = @Schema(implementation = Order.class))),
22             @ApiResponse(responseCode = "400",
23                 description = "The request parameters are invalid",
24                 content = @Content(
25                     mediaType = "application/json")),
26             @ApiResponse(responseCode = "404",
27                 description = "A Jeep component was not found with the input criteria",
28                 content = @Content(
29                     mediaType = "application/json")),
30             @ApiResponse(responseCode = "500",
31                 description = "An unplanned error occurred",
32                 content = @Content(
33                     mediaType = "application/json"))
34         },
35         parameters = {
36             @Parameter(name = "orderRequest",
37                 required = true,
38                 description = "The order as JSON"),
39         }
40     )
41     // @formatter: on
42     Order createOrder(@Valid @RequestBody OrderRequest orderRequest);
43 }
  
```

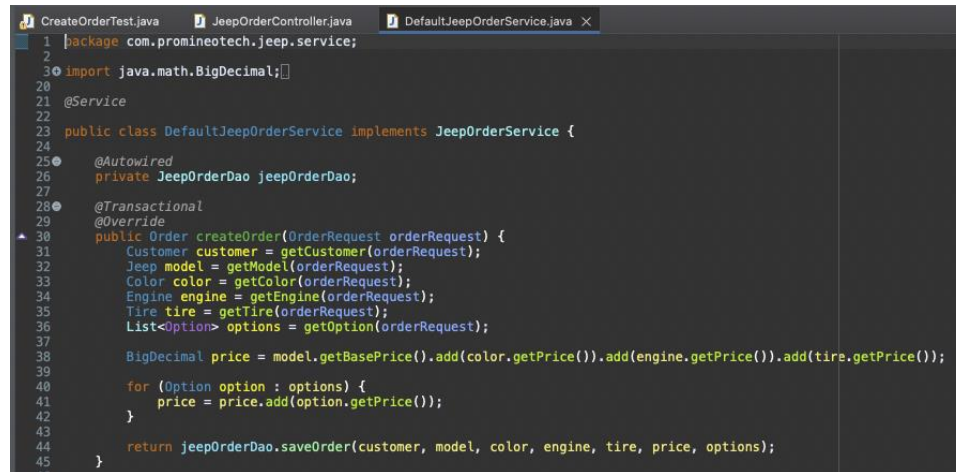
- Create a class that implements JeepOrderController named DefaultJeepOrderController.
  - Add `@RestController` as a class-level annotation.
  - Add a log line to the implementing controller method showing the input request body (orderRequest)
  - Run the test to show a red status bar. Produce a screenshot that shows the test method, the log line, and the red JUnit status bar.
- 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).
- Add the class-level annotation `@Validated` to the JeepOrderController interface.
- Add Bean Validation annotations to the OrderRequest class as shown in the video.
  - Use these annotations for String types:
    - `@NotNull`
    - `@Length(max = 30)`
    - `@Pattern(regexp = "[\\w\\s]*")`
  - Use these annotations for integer types:
    - `@Positive`
    - `@Min(2)`
    - `@Max(4)`
  - Add `@NotNull` to the enum type.

- 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.

- Produce a screenshot of this class with the annotations.



```

1 package com.promineotech.jeep.service;
2
3 import java.math.BigDecimal;
4
5 @Service
6 public class DefaultJeepOrderService implements JeepOrderService {
7
8     @Autowired
9     private JeepOrderDao jeepOrderDao;
10
11     @Transactional
12     @Override
13     public Order createOrder(OrderRequest orderRequest) {
14         Customer customer = getCustomer(orderRequest);
15         Jeep model = getModel(orderRequest);
16         Color color = getColor(orderRequest);
17         Engine engine = getEngine(orderRequest);
18         Tire tire = getTire(orderRequest);
19         List<Option> options = getOption(orderRequest);
20
21         BigDecimal price = model.getBasePrice().add(color.getPrice()).add(engine.getPrice()).add(tire.getPrice());
22
23         for (Option option : options) {
24             price = price.add(option.getPrice());
25         }
26
27         return jeepOrderDao.saveOrder(customer, model, color, engine, tire, price, options);
28     }
29 }

```

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

Order createOrder(OrderRequest orderRequest);

- Call the createOrder method from the controller and return the value returned by the service.
  - Add a log line in the createOrder method and log the orderRequest parameter.
  - 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).
- 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.
- 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.
- **\*\*\* 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.**
- 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.

- 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.

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

- 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.
- Calculate the price, including all options.
- 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);

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

```

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

```

- 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.

- 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.



- Produce a screenshot of the saveOrder method.

```

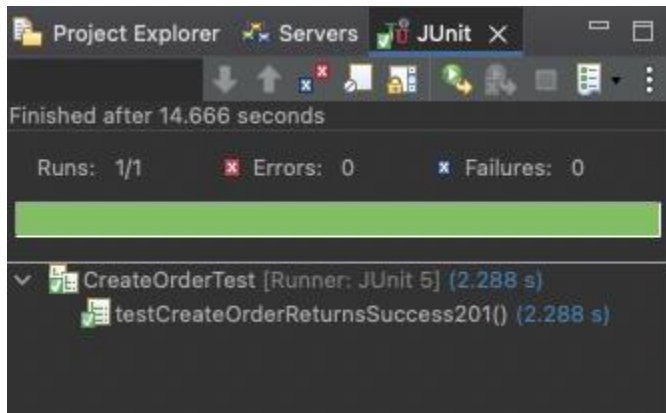
37
38  @Override
39  public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price,
40                          List<Option> options) {
41      SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);
42
43      KeyHolder keyHolder = new GeneratedKeyHolder();
44      jdbcTemplate.update(params.sql, params.source, keyHolder);
45
46      Long orderPK = keyHolder.getKey().longValue();
47      saveOptions(options, orderPK);
48
49      return Order.builder().orderPK(orderPK).customer(customer).model(jeep).color(color).engine(engine).tire(tire)
50                      .options(options).price(price).build();
51  }
52

```

- 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/Asosa809/Saving-POST-Data>