URL to GitHub Repository: https://github.com/tmccarthy91/Week-13-Assignments

URL to Public Link of your Video: https://youtu.be/MSXJnQ_ybTI

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 functionality into your Video when you see: 
    - Create a video showcasing your work:
        - In this video: record and present your project verbally while showing the results of the working project. Don't forget to include the requested functionality, indicated by: 
        - 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 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.

Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When something should be added into your video submission, look for the icon: 💻

Note: 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.jeep.controller` package.

Add the Spring Boot Test annotations: `@SpringBootTest`, `@ActiveProfiles`, and `@Sql`. They should have the same parameters as the test created in weeks 1 and 2.

Create a test method (annotated with `@Test`) named `testCreateOrderReturnsSuccess201`.

In the test class, create a method named `createOrderBody`. This method returns a type of String. In this method, return a JSON object with the IDs that you picked in Step 1a and 1b. 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/.

In your video show the `createOrderBody()` 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.jeep.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.CREATE
D);

assertThat(response.getBody()).isNotNull();


Order order = response.getBody();

assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORIS
ON_LINA");

assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WR
ANGLER);

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);
```

In your video, show the test 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.

In your video, show the finished `JeepOrderController` interface showing no compile errors. 💻

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. In your video, show 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:

i) `@NotNull`

ii) `@Length(max = 30)`

iii) `@Pattern(regexp = "[\\w\\s]*")`

Use these annotations for integer types:

i) `@Positive`

ii) `@Min(2)`

iii) `@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.

In you video, show this class with the annotations. 💻

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. In the video, show 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 jeepOrderDao.saveOrder method from the jeepOrderSalesService.createOrder service. Show in your video the jeepOrderSalesService.createOrder method. 💻

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.

In your video, show the saveOrder method. 💻

Run the integration test in `CreateOrderTest`. In your video, show the test method

that shows the green JUnit status bar, the console output, and the test class.