



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

URL to Public Link of your Video: <https://youtu.be/IdImxIgl22U>

Instructions :

1. Follow the [Exercises](#) 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 there is something that should be highlighted in your video submission, you will see this: 

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.

Here's a hint: make sure you are running a version of Java that is 11+. To get the version, open a Windows Command Prompt window or a Mac Terminal window and type `java -version`. If you need to upgrade, go here: <https://docs.aws.amazon.com/corretto/latest/corretto-11-ug/downloads-list.html>. Pick the .msi installer version (Windows) or the .pkg version (Mac).

Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

Create a Maven project named JeepSales as described in the video.

In Spring Tool Suite, click the "File" menu. Select "New/Project...". In the popup, expand "Maven" and select "Maven Project". Click "Next".

Check "Create a simple project (skip archetype selection)". Click "Next".

Enter the following:

Group Id	com.promineotech
Artifact Id	jeep-sales

a)

Click "Finish".

Navigate to the Spring Initializr (<https://start.spring.io/>).

Confirm the following settings:

Project	Maven Project
Language	Java
Spring Boot	Select the latest stable version (not SNAPSHOT or RC)
Group	com.promineotech
Artifact	jeep-sales
Name	jeep-sales
Description	Jeep Sales
Package name	com.promineotech
Packaging	Jar
Java	11 (or whatever your version is)

Add the dependencies from the Initializr:

Web

Devtools

Lombok

Click "Explore" at the bottom of the page.

Click "Copy" to copy the pom.xml generated by the Initializr to the clipboard.

In **Spring Tool Suite**, open pom.xml (in the project root directory). Select all the text in the editor and replace it with the XML copied to the clipboard in the prior step.

Navigate to <https://mvnrepository.com/>. Search for springdoc-openapi-ui. Select the latest version and add the entry to the POM file in the <dependencies> section.

Create a package in src/main/java named com.promineotech.jeepp. In this package:

Create a Java class with a main method named JeepSales.

Add a class-level annotation: @SpringBootApplication and the import statement.

In the main() method, add a call to SpringApplication.run();. Use JeepSales.class as the first parameter, and the args parameter that was passed into the main() method as the second. The entire class should look like this:

```
package com.promineotech.jeepp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class JeepSales {

    public static void main(String[] args) {
        SpringApplication.run(JeepSales.class, args);
    }
}
```

Refer to README.docx in the supplied project resources. Copy all files in the Files folder in the resources to your project as described in the README. **Do not copy the files in the Entity or Source folders at this time.**

Load the files that were added: right-click on the project in Package Explorer and

select "Refresh".

Update the project with the new POM dependencies: right-click on the project in Package Explorer, select "Maven/Update Project". When the "Update Maven Project" panel appears, click "OK".

Using the MySQL Workbench or MySQL command line client (CLI), create a database named "jeep".

Using DBeaver, or the MySQL client of choice, load the supplied .sql files (V1.0__Jeep_Schema.sql, and V1.1__Jeep_Data.sql) into the MySQL database to create the tables and populate them with data. These files are found in the project folder src/test/resources/flyway/migrations.

Create a new package in src/test/java named com.promineotech.jeep.controller. Create a Spring Boot integration test named FetchJeepTest using the techniques shown in the video.

Add the @SpringBootTest, @ActiveProfiles, and @Sql annotations as described in the video.

The class must not be public. It should have package-level access (i.e., not public, private, or protected).

The video extended FetchJeepTestSupport, but you don't need to do that for the homework. Just put everything in FetchJeepTest. It should look like this:

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@Sql(scripts = {
    "classpath:flyway/migrations/V1.0__Jeep_Schema.sql",
    "classpath:flyway/migrations/V1.1__Jeep_Data.sql"},
    config = @SqlConfig(encoding = "utf-8"))
class FetchJeepTest {
}
```

Create a test method in FetchJeepTest. The method must have the following method signature:

```
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()
```

Inject a TestRestTemplate in the test class. Name the variable restTemplate. Inject the port used in the test using the @LocalServerPort annotation. Name the variable serverPort. The variables and annotations should look like this:

```
@Autowired
```

```
private TestRestTemplate restTemplate;
```

```
@LocalServerPort
```

```
private int serverPort;
```

Create a new package in src/main/java named com.promineotech.jeepp.entity. In that package, create an enum named JeepModel. Add all the jeep models from the model_id column in the models table in the database. You can use this query in dBeaver: SELECT DISTINCT model_id FROM models.

Create a Jeep class in the com.promineotech.jeepp.entity package. Add the columns from the models table into this class as instance variables. Annotate the class with the Lombok annotations @Data, @Builder (and optionally both @NoArgsConstructor and @AllArgsConstructor). Note that modelId should be of type JeepModel and basePrice should be of type BigDecimal. The class should look like this (remember to add the appropriate import statements):

```
@Data
```

```
@Builder
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class Jeep {  
    private Long modelPK;  
    private JeepModel modelId;  
    private String trimLevel;  
    private int numDoors;  
    private int wheelSize;  
    private BigDecimal basePrice;  
}
```

In the supplied resources, copy all files in the Entities folder to the src/main/java/com/-promineotech/jepp/entity folder. **Do not copy anything from the Source folder at this time.**

Back in the test method that you were writing, create local variables for JeepModel, trim, and uri. Set them appropriately like this:

Variable Type	Variable Name	Variable Value
JeepModel	model	JeepModel.WRANGLER
String	trim	"Sport"
String	uri	String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);

1)

- a) Send an HTTP request to the REST service that passes a JeepModel and trim level as URI parameters (as shown in the video). Use this method call:

```
ResponseEntity<List<Jeep>> response = restTemplate.exchange(uri,
    HttpMethod.GET, null, new ParameterizedTypeReference<>()
    {});
```


Make sure to use the import `java.util.List` and `org.springframework.http.HttpMethod`.

- a) Using [AssertJ](#), test that the response that comes back from the server is 200 (success) – or as is shown in the video: `HttpStatus.OK`. The code should look like this:

```
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
```

Use the import statements:

```
import static org.assertj.core.api.Assertions.assertThat;
```

- a) In your video show the completed test class. 

- 1) In `src/main/java`, create a new package `com.promineotech.jeep.controller`. In this package, create an interface named `JeepSalesController`.

Add the class-level annotation `@RequestMapping("/jeeps")`.

Add the `fetchJeeps` method in a controller interface with the following signature:

```
List<Jeep> fetchJeeps(JeepModel model, String trim);
```

Make sure you use the `List` from `java.util.List`.

Add OpenAPI documentation to document the four possible outcomes: 200 (success), 400 (bad input), 404 (not found) and 500 (unplanned error) as shown in the video.


Add the parameter annotations in the OpenAPI documentation to describe the

model and trim parameters.

Add the `@GetMapping` annotation and the `@ResponseStatus(code = HttpStatus.OK)` annotation as method-level annotations to the `fetchJeeps` method.

Add the `@RequestParam` annotations to the parameters as described in the video. The interface should look like this (omitting the OpenAPI annotations):

```
@RequestMapping("/jeeps")
public interface JeepSalesController {
    @GetMapping
    @ResponseStatus(code = HttpStatus.OK)
    List<Jeep> fetchJeeps(@RequestParam JeepModel model,
        @RequestParam String trim);
}
```

In your video, show the interface and OpenAPI documentation. 

Add the controller implementation class named `DefaultJeepSalesController`. Don't forget the `@RestController` annotation.

Run the application within the IDE and show the resulting OpenAPI (Swagger) documentation produced in the browser. In your video, make sure to show the OpenAPI (Swagger) documentation produced in the browser, and show all four possible outcomes. 