


Web API Design with Spring Boot Week 1 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.

Here's a hint: make sure you are running a version of Java that is 11+. To get the version, open a Windows command 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:

- 1) Create a Maven project named `JeepSales` as described in the video.

- a) In Spring Tool Suite, click the "File" menu. Select "New/Project...". In the popup, expand "Maven" and select "Maven Project". Click "Next".
- b) Check "Create a simple project (skip archetype selection)". Click "Next".
- c) Enter the following:

Group Id	com.promineotech
Artifact Id	jeep-sales

Click "Finish".

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

- a) 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

- b) Add the dependencies from the Initializr:
 - i) Web
 - ii) Devtools
 - iii) Lombok
- c) Click "Explore" at the bottom of the page.
- d) Click "Copy" to copy the pom.xml generated by the Initializr to the clipboard.

- 3) 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.
- 4) 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.
- 5) Create a package in src/main/java named com.promineotech.jeeep. In this package:
 - a) Create a Java class with a main method named JeepSales.
 - b) Add a class-level annotation: @SpringBootApplication and the import statement.
 - c) 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.jeeep;

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);
    }
}
```

- 6) 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.**
 - a) Load the files that were added: right-click on the project in Package Explorer and select "Refresh".
 - b) 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".
- 7) Using the MySQL Workbench or MySQL command line client (CLI), create a database named "jeeep".
- 8) 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.
- 9) Create a new package in src/test/java named com.promineotech.jeeep.controller. Create a Spring Boot integration test named FetchJeepTest using the techniques shown in the video.

- a) Add the `@SpringBootTest`, `@ActiveProfiles`, and `@Sql` annotations as described in the video.
- b) The class must not be public. It should have package-level access (i.e., not public, private, or protected).
- c) 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 {
}
```

- d) Create a test method in `FetchJeepTest`. The method must have the following method signature:

```
void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied()
```

- e) 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;
```

- 10) Create a new package in `src/main/java` named `com.promineotech.jeep.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`.
- 11) Create a `Jeep` class in the `com.promineotech.jeep.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;
}

```

- 12) In the supplied resources, copy all files in the Entities folder to the src/main/java/com/-promineotech/jeep/entity folder. **Do not copy anything from the Source folder at this time.**
- 13) 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);

- 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.

- b) 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);


```

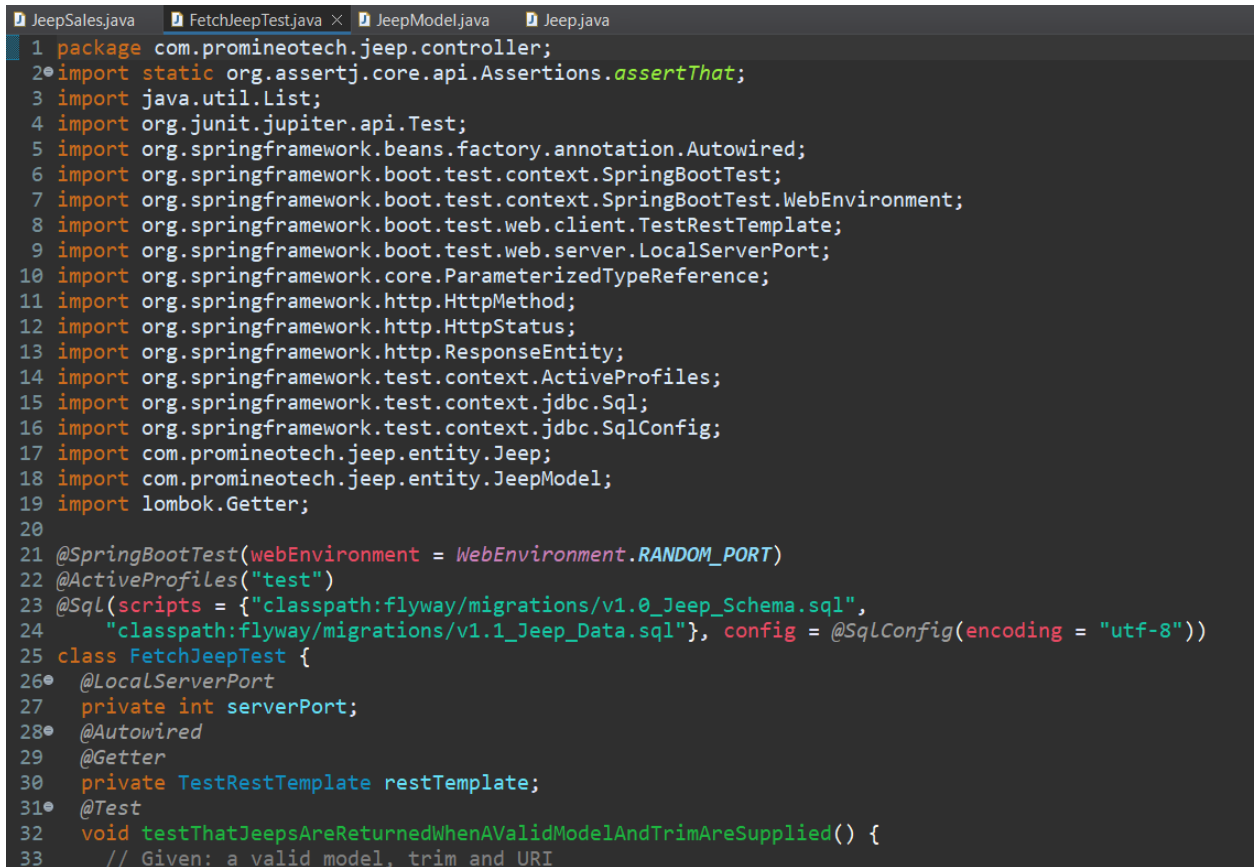
Use the import statements:

```

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

```

c) Produce a screenshot showing the completed test class. 



```
1 package com.promineotech.jeep.controller;
2 import static org.assertj.core.api.Assertions.assertThat;
3 import java.util.List;
4 import org.junit.jupiter.api.Test;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
8 import org.springframework.boot.test.web.client.TestRestTemplate;
9 import org.springframework.boot.test.web.server.LocalServerPort;
10 import org.springframework.core.ParameterizedTypeReference;
11 import org.springframework.http.HttpMethod;
12 import org.springframework.http.HttpStatus;
13 import org.springframework.http.ResponseEntity;
14 import org.springframework.test.context.ActiveProfiles;
15 import org.springframework.test.context.jdbc.Sql;
16 import org.springframework.test.context.jdbc.SqlConfig;
17 import com.promineotech.jeep.entity.Jeep;
18 import com.promineotech.jeep.entity.JeepModel;
19 import lombok.Getter;
20
21 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
22 @ActiveProfiles("test")
23 @Sql(scripts = {"classpath:flyway/migrations/v1.0_Jeep_Schema.sql",
24               "classpath:flyway/migrations/v1.1_Jeep_Data.sql"}, config = @SqlConfig(encoding = "utf-8"))
25 class FetchJeepTest {
26     @LocalServerPort
27     private int serverPort;
28     @Autowired
29     @Getter
30     private TestRestTemplate restTemplate;
31     @Test
32     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
33         // Given: a valid model, trim and URI
```

```

14 import org.springframework.test.context.ActiveProfiles;
15 import org.springframework.test.context.jdbc.Sql;
16 import org.springframework.test.context.jdbc.SqlConfig;
17 import com.promineotech.jee.entity.Jee;
18 import com.promineotech.jee.entity.JeeModel;
19 import lombok.Getter;
20
21 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
22 @ActiveProfiles("test")
23 @Sql(scripts = {"classpath:flyway/migrations/v1.0_Jee_Schema.sql",
24               "classpath:flyway/migrations/v1.1_Jee_Data.sql"}, config = @SqlConfig(encoding = "utf-8"))
25 class FetchJeeTest {
26     @LocalServerPort
27     private int serverPort;
28     @Autowired
29     @Getter
30     private TestRestTemplate restTemplate;
31     @Test
32     void testThatJeepsAreReturnedWhenAValidModelAndTrimAreSupplied() {
33         // Given: a valid model, trim and URI
34         JeeModel model = JeeModel.WRANGLER;
35         String trim = "Sport";
36         String uri = String.format("http://localhost:%d/jeeps?model=%s&trim=%s", serverPort, model, trim);
37
38         // When: a connection is made to the URI
39         getRestTemplate().getForEntity(uri, Jee.class);
40
41         // Then: a success (OK - 200) status code is returned
42         ResponseEntity<List<Jee>> response = restTemplate.exchange
43             (uri, HttpMethod.GET, null, new ParameterizedTypeReference<>() {});
44         assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
45     }
46 }

```

14) In src/main/java, create a new package com.promineotech.jee.controller. In this package, create an interface named JeepSalesController.

- a) Add the class-level annotation @RequestMapping("/jeeps").
- b) Add the fetchJeeps method in a controller interface with the following signature:
List<Jee> fetchJeeps(JeeModel model, String trim);
Make sure you use the List from java.util.List.
- c) 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.
- d) Add the parameter annotations in the OpenAPI documentation to describe the model and trim parameters.
- e) Add the @GetMapping annotation and the @ResponseStatus(code = HttpStatus.OK) annotation as method-level annotations to the fetchJeeps method.
- f) 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

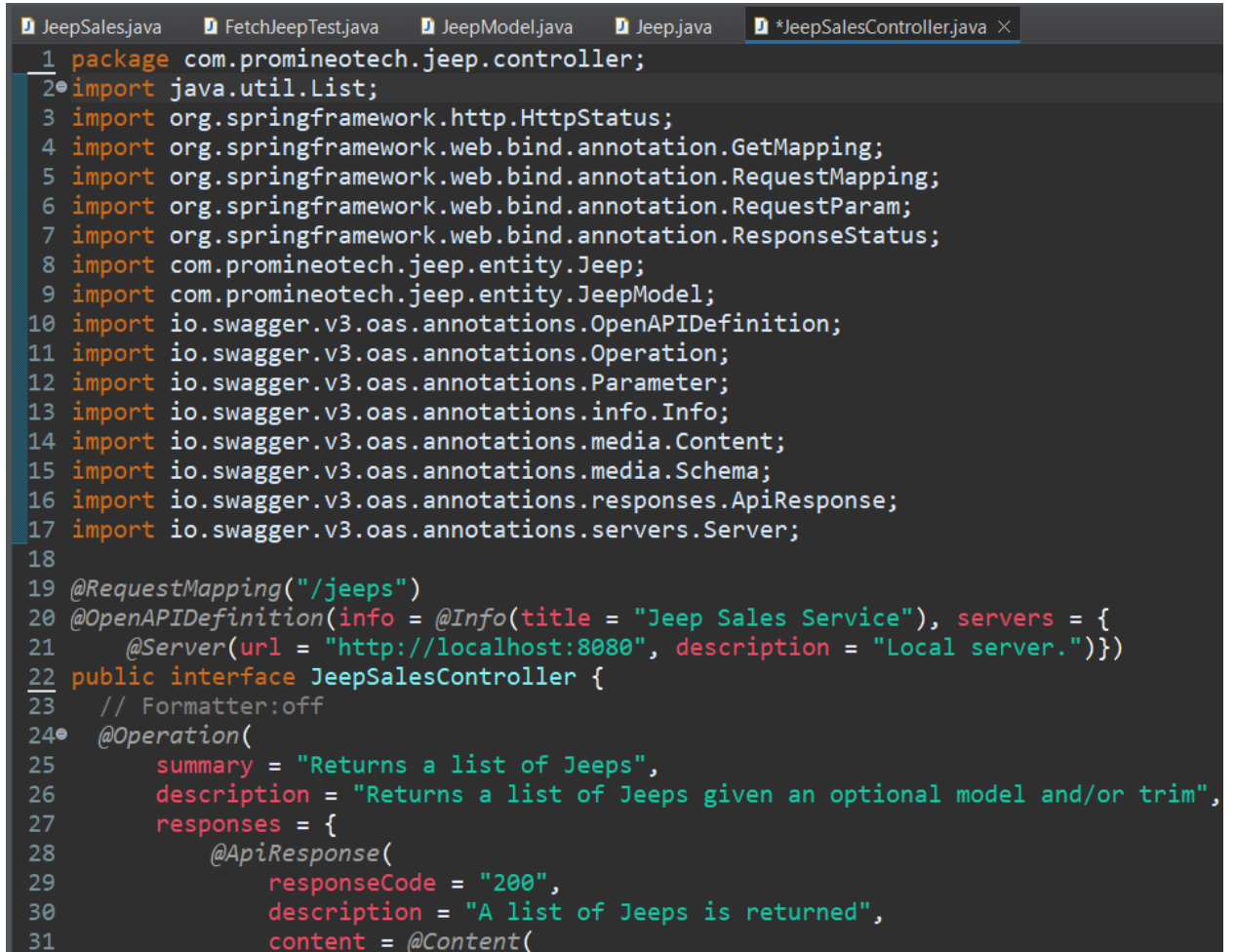
```

```

    @ResponseStatus(code = HttpStatus.OK)
    List<Jeep> fetchJeeps(@RequestParam JeepModel model,
        @RequestParam String trim);
}

```

g) Produce a screenshot showing the interface and OpenAPI documentation. 



```

1 package com.promineotech.jeep.controller;
2 import java.util.List;
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestParam;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8 import com.promineotech.jeep.entity.Jeep;
9 import com.promineotech.jeep.entity.JeepModel;
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.responses.ApiResponse;
17 import io.swagger.v3.oas.annotations.servers.Server;
18
19 @RequestMapping("/jeeps")
20 @OpenAPIDefinition(info = @Info(title = "Jeep Sales Service"), servers = {
21     @Server(url = "http://localhost:8080", description = "Local server.")})
22 public interface JeepSalesController {
23     // Formatter:off
24     @Operation(
25         summary = "Returns a list of Jeeps",
26         description = "Returns a list of Jeeps given an optional model and/or trim",
27         responses = {
28             @ApiResponse(
29                 responseCode = "200",
30                 description = "A list of Jeeps is returned",
31                 content = @Content(

```



```
JeepSales.java  FetchJeepTest.java  JeepModel.java  Jeep.java  *JeepSalesController.java X
31         content = @Content(
32             MediaType = "application/json",
33             schema = @Schema(implementation = Jeep.class)),
34     @ApiResponse(
35         responseCode = "400",
36         description = "The request parameters are invalid",
37         content = @Content(MediaType = "application/json")),
38     @ApiResponse(
39         responseCode = "404",
40         description = "No Jeeps were found with the input criteria",
41         content = @Content(MediaType = "application/json")),
42     @ApiResponse(
43         responseCode = "500",
44         description = "An unplanned error occurred.",
45         content = @Content(MediaType = "application/json"))
46
47     },
48     parameters = {
49         @Parameter(
50             name = "model",
51             allowEmptyValue = false,
52             required = false,
53             description = "The Modelname (i.e., 'WRANGLER')"),
54         @Parameter(
55             name = "trim",
56             allowEmptyValue = false,
57             required = false,
58             description = "The trim level (i.e., 'Sport')")
59     }
60 )
61
62 // formatter:on
63 @GetMapping
64 @ResponseStatus(code = HttpStatus.OK)
65 List<Jeep> fetchJeeps(@RequestParam JeepModel model,
66                     @RequestParam String trim);
67 }
68
```

- 15) Add the controller implementation class named DefaultJeepSalesController. Don't forget the @RestController annotation.
- 16) Run the application within the IDE and show the resulting OpenAPI (Swagger) documentation produced in the browser. Produce a screenshot of the documentation showing

all four possible outcomes. 🖥️

200	A list of Jeeps is returned	No links
	<div>Media type application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "modelPK": 0, "modelId": "WRANGLER", "trimLevel": "string", "numDoors": 0, "wheelSize": 0, "basePrice": 0 }</pre></div>	
400	The request parameters are invalid	No links
	<div>Media type application/json</div>	
404	No Jeeps were found with the input criteria	No links
	<div>Media type application/json</div>	
500	An unplanned error occurred.	No links
	<div>Media type application/json</div>	

Screenshots of Code:

```
JeepSales.java x FetchJeepTe... JeepSalesCo... DefaultJeepS... OrderReques... Tire.java
1 package com.promineotech.jeep;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class JeepSales {
8
9
10 public static void main(String[] args) {
11     SpringApplication.run(JeepSales.class, args);
12 }
13 }
```

```
JeepSales.java  FetchJeepTe...  JeepSalesCo...  DefaultJeepS... ×  OrderReques...  Tire.java
1 package com.promineotech.jeep.controller;
2
3 import java.util.List;
4 import org.springframework.web.bind.annotation.RestController;
5 import com.promineotech.jeep.entity.Jeep;
6 import com.promineotech.jeep.entity.JeepModel;
7
8 @RestController
9 public class DefaultJeepSalesController implements JeepSalesController {
10
11     @Override
12     public List<Jeep> fetchJeeps(JeepModel model, String trim) {
13         return null;
14     }
15
16 }
17
```

```
JeepSales.java  FetchJeepTe...  JeepModel.java  Jeep.java ×  JeepSalesCo...  DefaultJeepS...  »4
1 package com.promineotech.jeep.entity;
2
3 import java.math.BigDecimal;
4 import lombok.AllArgsConstructor;
5 import lombok.Builder;
6 import lombok.Data;
7 import lombok.NoArgsConstructor;
8
9 @Data
10 @Builder
11 @NoArgsConstructor
12 @AllArgsConstructor
13
14 public class Jeep {
15     private Long modelPK;
16     private JeepModel modelId;
17     private String trimLevel;
18     private int numDoors;
19     private int wheelSize;
20     private BigDecimal basePrice;
21 }
```

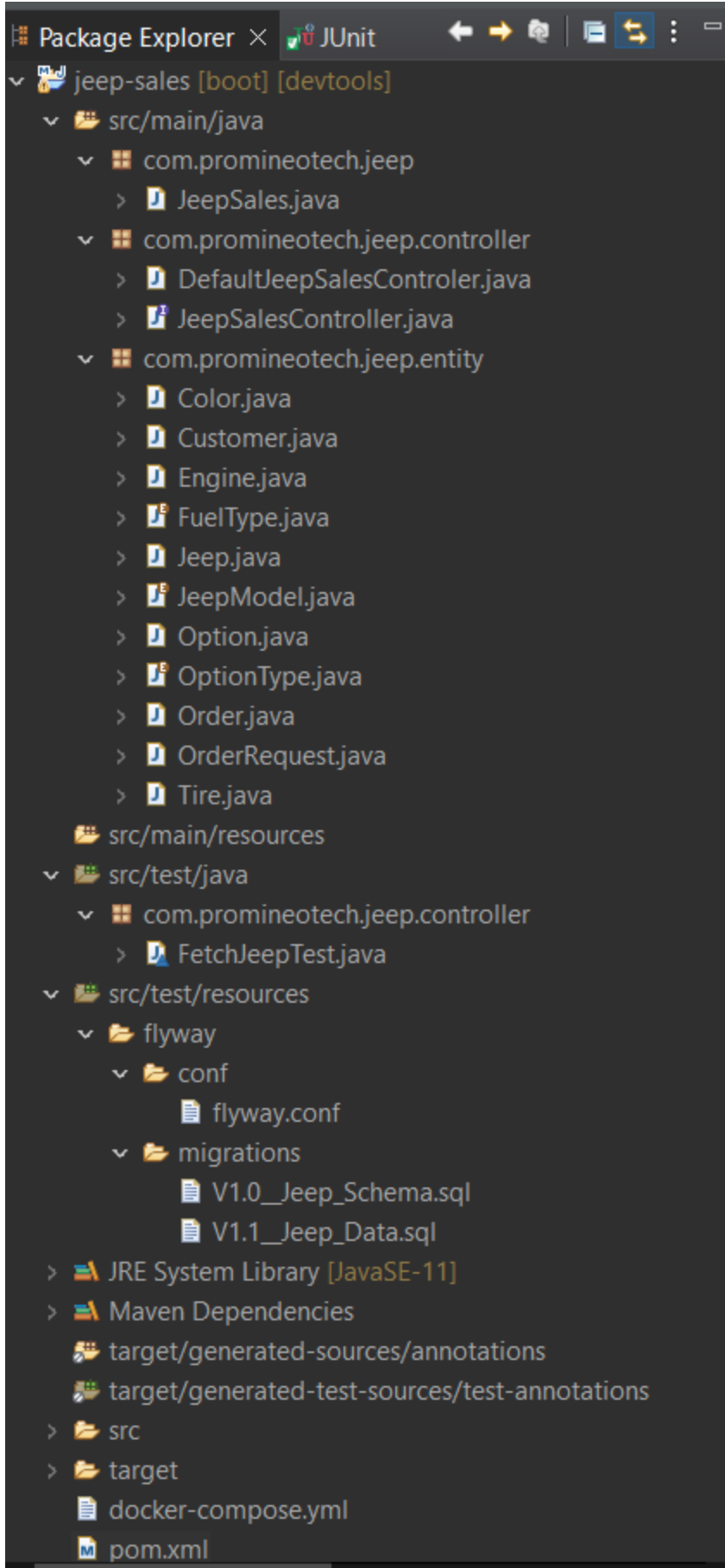
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     https://maven.apache.org/xsd/maven-4.0.0.xsd">
6   <modelVersion>4.0.0</modelVersion>
7
8   <parent>
9     <groupId>org.springframework.boot</groupId>
10    <artifactId>spring-boot-starter-parent</artifactId>
11    <version>2.7.2</version>
12    <relativePath /> <!-- lookup parent from repository -->
13  </parent>
14  <groupId>com.promineotech</groupId>
15  <artifactId>jeepp-sales</artifactId>
16  <version>0.0.1-SNAPSHOT</version>
17  <name>jeepp-sales</name>
18  <description>Demo project for Spring Boot</description>
19  <properties>
20    <java.version>11</java.version>
21  </properties>
22  <dependencies>
23    <dependency>
24      <groupId>org.springdoc</groupId>
25      <artifactId>springdoc-openapi-ui</artifactId>
26      <version>1.6.9</version>
27    </dependency>
28
29    <dependency>
30      <groupId>org.springframework.boot</groupId>
31      <artifactId>spring-boot-starter-web</artifactId>
32    </dependency>

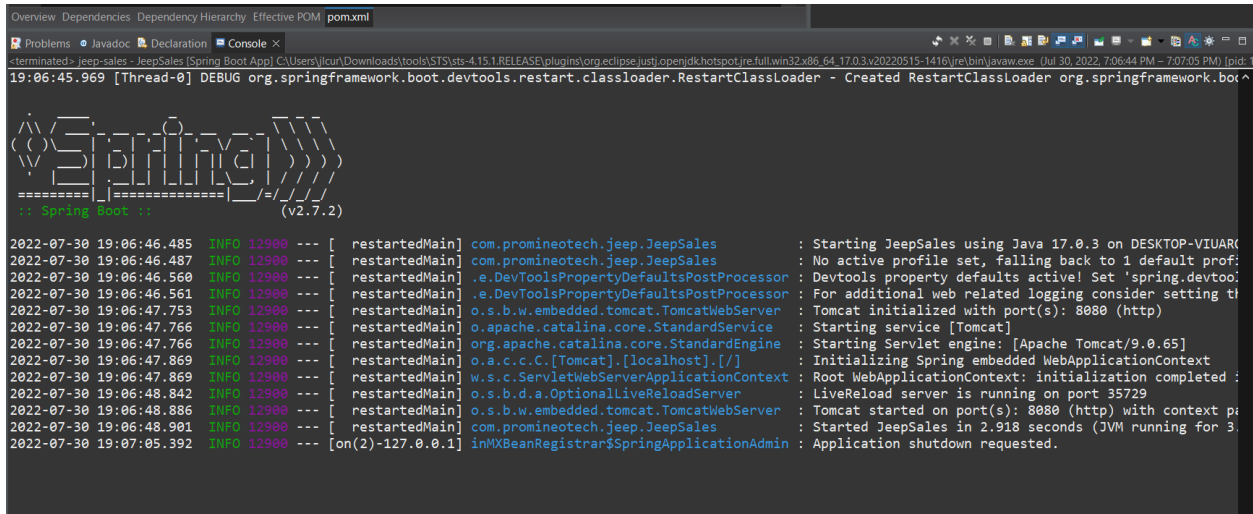
```

```
JeepSales.java  FetchJeepTe...  Jeep.java  JeepSalesCo...  DefaultJeepS...  jeep-sales/p... X
33
34  <dependency>
35    <groupId>org.springframework.boot</groupId>
36    <artifactId>spring-boot-devtools</artifactId>
37    <scope>runtime</scope>
38    <optional>true</optional>
39  </dependency>
40  <dependency>
41    <groupId>org.projectlombok</groupId>
42    <artifactId>lombok</artifactId>
43    <optional>true</optional>
44  </dependency>
45  <dependency>
46    <groupId>org.springframework.boot</groupId>
47    <artifactId>spring-boot-starter-test</artifactId>
48    <scope>test</scope>
49  </dependency>
50 </dependencies>
51
52 <build>
53   <plugins>
54     <plugin>
55       <groupId>org.springframework.boot</groupId>
56       <artifactId>spring-boot-maven-plugin</artifactId>
57       <configuration>
58         <excludes>
59           <exclude>
60             <groupId>org.projectlombok</groupId>
61             <artifactId>lombok</artifactId>
62           </exclude>
63         </excludes>
64       </configuration>
65     </plugin>
66   </plugins>
67 </build>
68
69 </project>
```

Overview · Dependencies · Dependency Hierarchy · Effective POM · pom.xml



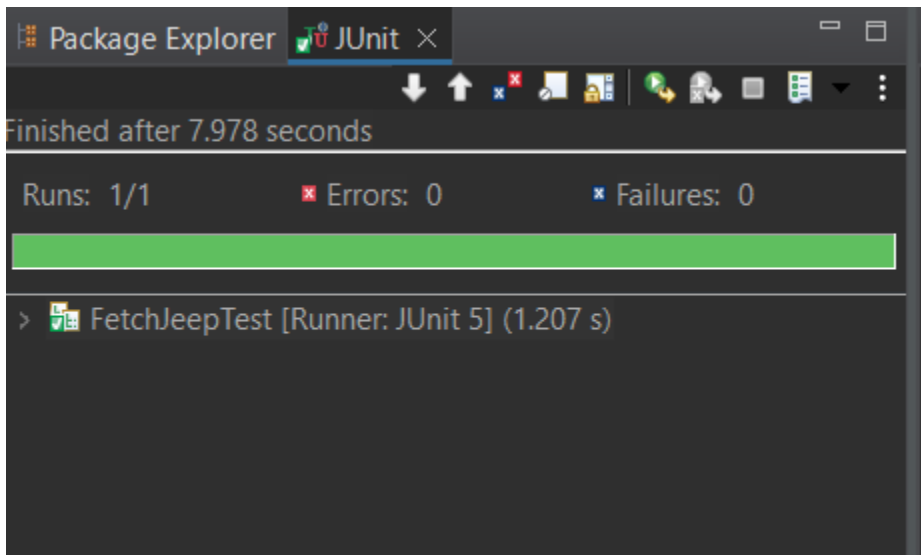
Screenshots of Running Application:



The screenshot shows the Eclipse IDE console with the 'pom.xml' tab selected. The console output displays the following logs:

```
<terminated> JeepSales - JeepSales [Spring Boot App] C:\Users\jcurran\Downloads\tools\STS\sts-4.15.1.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\java.exe (Jul 30, 2022, 7:06:44 PM) [pid: 19:06:45.969 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.restart.classloader.RestartClassLoader (v2.7.2)

2022-07-30 19:06:46.485 INFO 12900 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Starting JeeppSales using Java 17.0.3 on DESKTOP-VIUA...
2022-07-30 19:06:46.487 INFO 12900 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : No active profile set, falling back to 1 default profi...
2022-07-30 19:06:46.560 INFO 12900 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtool...
2022-07-30 19:06:46.561 INFO 12900 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting th...
2022-07-30 19:06:47.753 INFO 12900 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-07-30 19:06:47.766 INFO 12900 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-07-30 19:06:47.766 INFO 12900 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-07-30 19:06:47.869 INFO 12900 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-07-30 19:06:47.869 INFO 12900 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed :
2022-07-30 19:06:48.842 INFO 12900 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-07-30 19:06:48.886 INFO 12900 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context pa...
2022-07-30 19:06:48.901 INFO 12900 --- [ restartedMain] com.promineotech.jeepp.JeeppSales : Started JeeppSales in 2.918 seconds (JVM running for 3...
2022-07-30 19:07:05.392 INFO 12900 --- [on(2)-127.0.0.1] inMXBeanRegistrar$SpringApplicationAdmin : Application shutdown requested.
```



The screenshot shows the Eclipse IDE Package Explorer with the 'JUnit' tab selected. The test results for 'FetchJeepTest' are displayed, showing a successful run.

Finished after 7.978 seconds

Runs: 1/1 ✖ Errors: 0 ✖ Failures: 0

> FetchJeepTest [Runner: JUnit 5] (1.207 s)

URL to GitHub Repository:

<https://github.com/juliecurran3/JeepSalesWeek1.git>