# 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.jeep.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.

```
65    }
66
67    String createOrderBody() {
68      return "{\n"
69          + "  \"customer\":\"STERN_TORO\",\n"
70          + "  \"model\":\"COMPASS\",\n"
71          + "  \"trim\":\"TRAILHAWK\",\n"
72          + "  \"doors\":4,\n"
73          + "  \"color\":\"OLIVE_GREEN\",\n"
74          + "  \"engine\":\"3_0_DIESEL\",\n"
75          + "  \"tire\":\"35_TOYO\",\n"
76          + "  \"options\":[\n"
77          + "    \"DOOR_QUAD_4\",\n"
78          + "    \"EXT_QUAD_ALUM_FRONT\",\n"
79          + "    \"EXT_WARN_WINCH\",\n"
80          + "    \"EXT_WARN_BUMPER_FRONT\",\n"
81          + "    \"EXT_WARN_BUMPER_REAR\",\n"
82          + "    \"EXT_ARB_COMPRESSOR\"\n"
83          + "  ]\n"
84          + "}\n"
85          + "";
86    }
87  }
```

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.jeep.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.

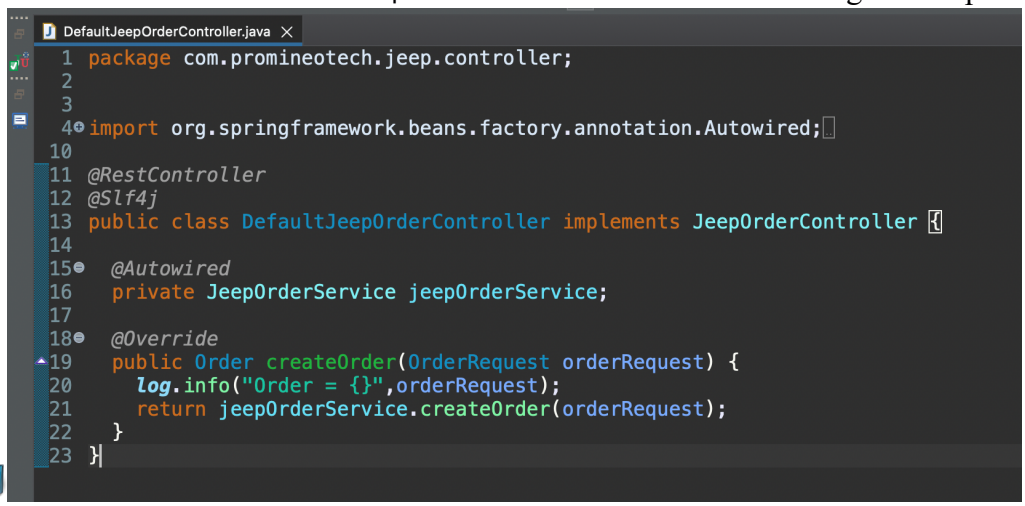
```
41
42  @Test
43  void testCreateOrderReturnsSuccess201() {
44
45    String body = createOrderBody();
46    String uri = String.format("http://localhost:%d/orders", serverPort);
47    HttpHeaders headers = new HttpHeaders();
48    headers.setContentType(MediaType.APPLICATION_JSON);
49    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
50    ResponseEntity<Order> response = restTemplate.exchange(uri,HttpMethod.POST, bodyEntity, Order.class);
51
52    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
53
54    assertThat(response.getBody()).isNotNull();
55
56    Order order = response.getBody();
57    assertThat(order.getCustomer().getCustomerId()).isEqualTo("STERN_TORO");
58    assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.COMPASS);
59    assertThat(order.getModel().getTrimLevel()).isEqualTo("Trailhawk");
60    assertThat(order.getModel().getNumDoors()).isEqualTo(4);
61    assertThat(order.getColor().getColorId()).isEqualTo("OLIVE_GREEN");
62    assertThat(order.getEngine().getEngineId()).isEqualTo("3_0_DIESEL");
63    assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
64    assertThat(order.getOptions()).hasSize(6);
65  }
66
```

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.

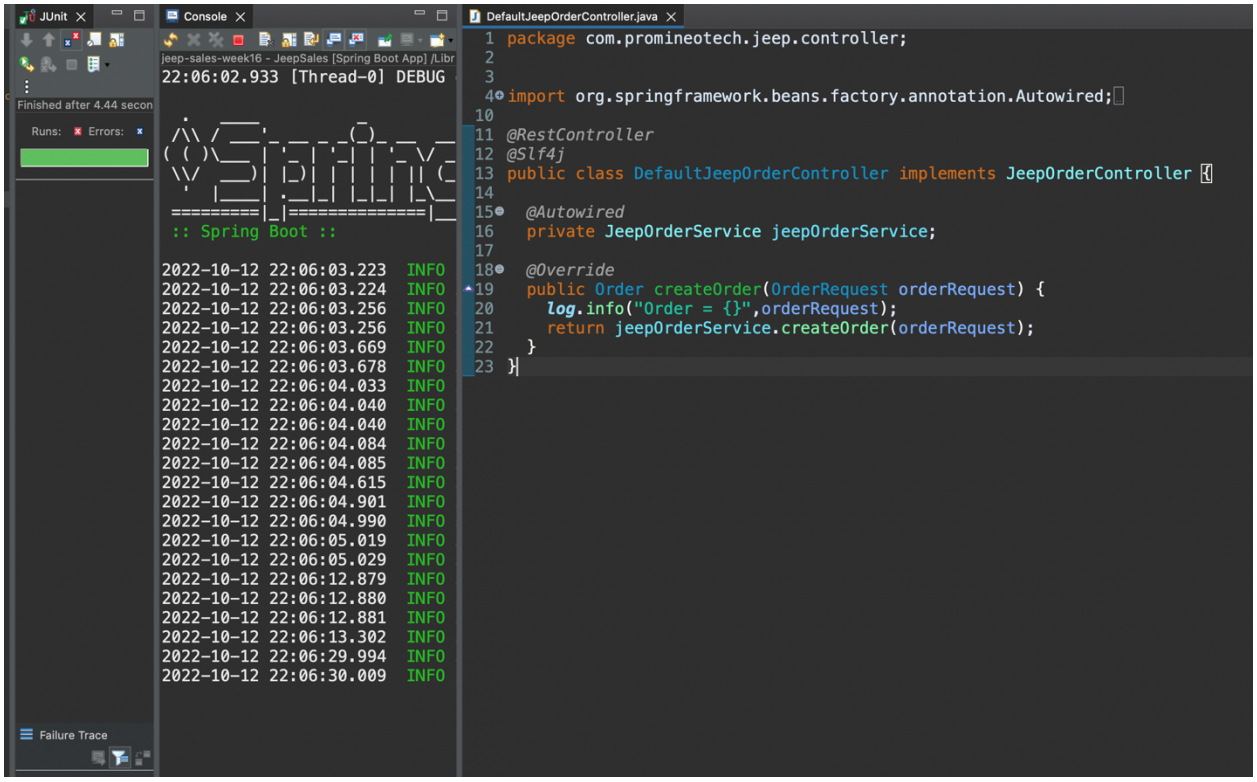



4) Create a class that implements `JeepOrderController` named `DefaultJeepOrderController`.

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. - Followed along with videos




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.


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.

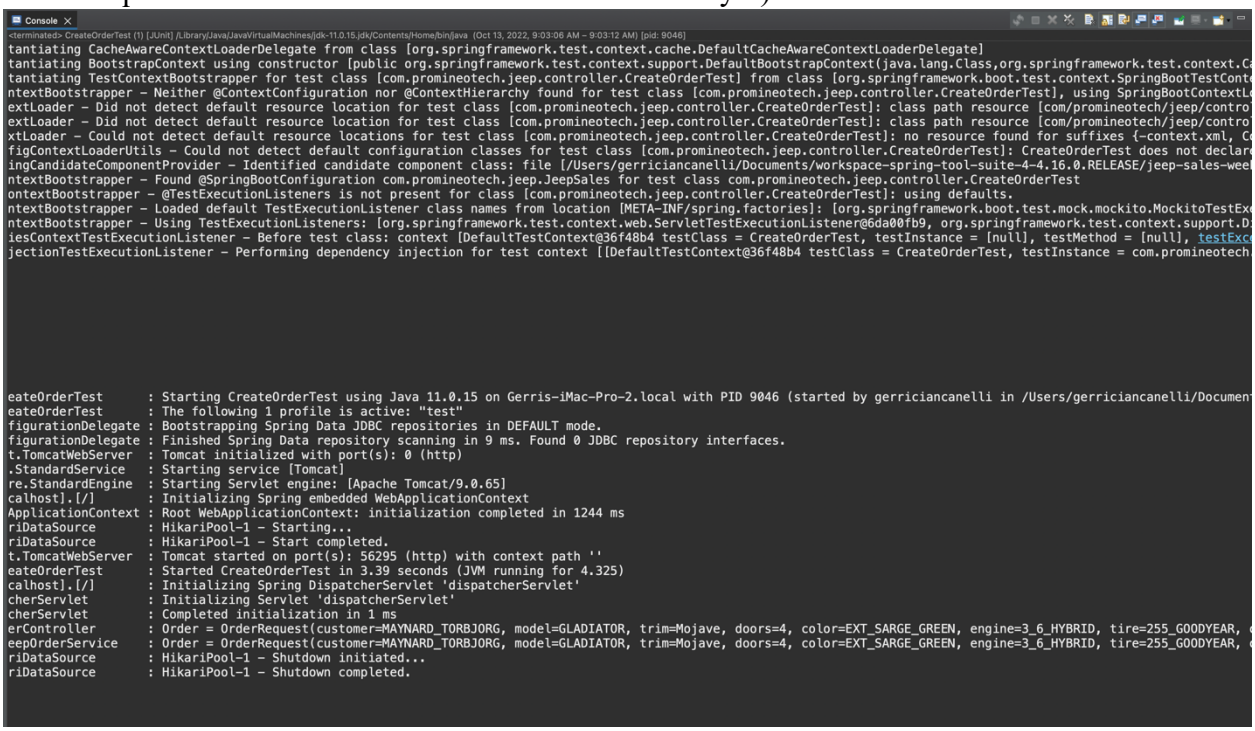
c) Create the `createOrder` method in the interface and implementing service. The method signature should look like this:

```
Order createOrder(OrderRequest orderRequest);
```

d) Call the `createOrder` method from the controller and return the value returned by the service.

e) Add a log line in the createOrder method and log the `orderRequest` parameter.

f) 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).




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) 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`.

13) 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.

14) 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.

15) 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 jeepOrder.Dao.saveOrder method from the jeepOrderSalesService.createOrder service. Produce a screenshot of the jeepOrderSalesService.createOrder method.

JeepOrderService.java

```java
package com.promineotech.jeep.service;

import com.promineotech.jeep.entity.Order;

public interface JeepOrderService {

  Order createOrder(OrderRequest orderRequest);

}
```

DefaultJeepOrderService.java | DefaultJeepOrderDao.java

```java
  @Transactional
  public Order createOrder(OrderRequest orderRequest) {
    log.info("Order = {}",orderRequest);

    Customer customer = getCustomer(orderRequest);
    Jeep jeep = getModel(orderRequest);
    Color color = getColor(orderRequest);
    Engine engine = getEngine(orderRequest);
    Tire tire = getTire (orderRequest);
    List<Option> options = getOption(orderRequest);

    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);
  }

  /**
```

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 `SqlParams` 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)
```

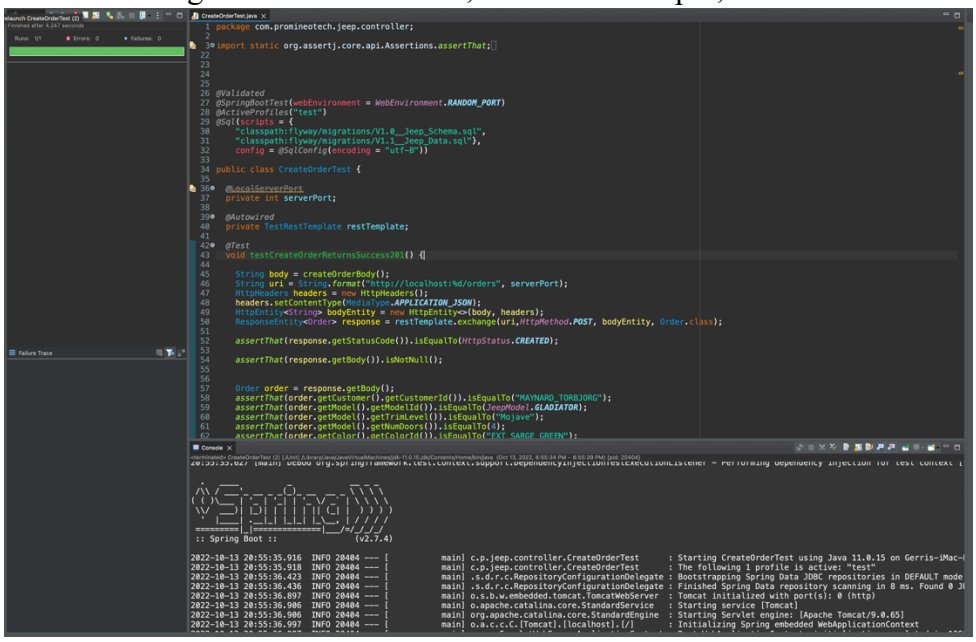
For each option in the `Options` list, call the supplied `generateInsertSql` method passing the parameters `option` and order primary key (`orderPK`). Call the `update` method on the `NamedParameterJdbcTemplate` object.

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.

```java
package com.promineotech.jeep.dao;

import java.math.BigDecimal;

public interface JeepOrderDao {
  List<Option> fetchOptions(List<String> optionIds);
  Optional<Customer> fetchCustomer(String customerId);
  Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
  Optional<Color> fetchColor(String colorId);
  Optional<Engine> fetchEngine(String engineId);
  Optional<Tire> fetchTire(String tireId);

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

}
```

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:

CreateOrderTest.java

```java
package com.promineotech.jeep.controller;

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




@Validated
@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"))

public class CreateOrderTest {

  @LocalServerPort
  private int serverPort;

  @Autowired
  private TestRestTemplate restTemplate;

  @Test
  void testCreateOrderReturnsSuccess201() {

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

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);

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

    Order order = response.getBody();
    assertThat(order.getCustomer().getCustomerId()).isEqualTo("STERN_TORO");
    assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.COMPASS);
    assertThat(order.getModel().getTrimLevel()).isEqualTo("Trailhawk");
    assertThat(order.getModel().getNumDoors()).isEqualTo(4);
    assertThat(order.getColor().getColorId()).isEqualTo("OLIVE_GREEN");
    assertThat(order.getEngine().getEngineId()).isEqualTo("3_0_DIESEL");
    assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
    assertThat(order.getOptions()).hasSize(6);
  }

  String createOrderBody() {
      //formatter:off
     return "{\n"
         + "  \"customer\":\"STERN_TORO\",\n"
         + "  \"model\":\"COMPASS\",\n"
         + "  \"trim\":\"TRAILHAWK\",\n"
         + "  \"doors\":4,\n"
         + "  \"color\":\"OLIVE_GREEN\",\n"
         + "  \"engine\":\"3_0_DIESEL\",\n"
         + "  \"tire\":\"35_TOYO\",\n"
         + "  \"options\":[\n"
         + "    \"DOOR_QUAD_4\",\n"
         + "    \"EXT_QUAD_ALUM_FRONT\",\n"
         + "    \"EXT_WARN_WINCH\",\n"
         + "    \"EXT_WARN_BUMPER_FRONT\",\n"
         + "    \"EXT_WARN_BUMPER_REAR\",\n"
         + "    \"EXT_ARB_COMPRESSOR\"\n"
         + "  ]\n"
         + "";

         //formatter:on
  }
}
```

CreateOrderTest.java | JeepOrderController.java

```java
package com.promineotech.jeep.controller;

import org.springframework.http.HttpStatus;

@Validated
@OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
        @Server(url = "http://localhost:8080", description = "Local server.")})

@RequestMapping("/orders")

public interface JeepOrderController {

      @Operation(
          summary = "Create an order for a Jeep",
          description = "Retruns the created Jeep",
          responses = {
              @ApiResponse(responseCode = "201",
                  description = "The created Jeep is returned",
                  content = @Content(mediaType = "application/json",
                  schema = @Schema(implementation = Jeep.class))),
              @ApiResponse(responseCode = "400",
                description = "The request parameters are invalid",
                content = @Content(mediaType = "application/json")),
              @ApiResponse(responseCode = "404",
                description = "A Jeep component was not found with the input criteria",
                content = @Content(mediaType = "application/json")),
              @ApiResponse(responseCode = "500",
                description = "An unplanned error occurred",
                content = @Content(mediaType = "application/json"))
          },
          parameters = {
              @Parameter(name = "orderRequest",
                required = true,
                description = "The order as JSON"),

          }
      )
      @PostMapping
      @ResponseStatus(code = HttpStatus.CREATED)
      Order createOrder(@RequestBody OrderRequest orderRequest);
    }
```

CreateOrderTest.java | JeepOrderController.java | DefaultJeepOrderController.java

```java
package com.promineotech.jeep.controller;


import org.springframework.beans.factory.annotation.Autowired;

@RestController
@Slf4j
public class DefaultJeepOrderController implements JeepOrderController {

  @Autowired
  private JeepOrderService jeepOrderService;

  @Override
  public Order createOrder(OrderRequest orderRequest) {
    log.info("Order = {}",orderRequest);
    return jeepOrderService.createOrder(orderRequest);
  }
}
```

CreateOrderTest.java | JeepOrderController.java | DefaultJeepOrderController.java | jeep-sales-week16_videowork/pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.4</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <groupId>com.promineotech</groupId>
  <artifactId>jeep-sales</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>jeep-sales</name>
  <description>Jeep Sales</description>

  <properties>
    <java.version>11</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <!-- Bean validation ================================-->
    <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>

  </dependency>

```

```
49    <!-- OpenAPI dependency ================================-->
50    <dependency>
51      <groupId>org.springdoc</groupId>
52      <artifactId>springdoc-openapi-ui</artifactId>
53      <version>1.6.11</version>
54  </dependency>
55
56
57      <dependency>
58        <groupId>org.springframework.boot</groupId>
59        <artifactId>spring-boot-starter-test</artifactId>
60        <scope>test</scope>
61      </dependency>
62
63       <dependency>
64        <groupId>org.springframework.boot</groupId>
65        <artifactId>spring-boot-starter-data-jdbc</artifactId>
66      </dependency>
67
68      <dependency>
69        <groupId>mysql</groupId>
70        <artifactId>mysql-connector-java</artifactId>
71      </dependency>
72  <!-- dependencies for tests ==============================-->
73  <dependency>
74      <groupId>com.h2database</groupId>
75      <artifactId>h2</artifactId>
76      <scope>test</scope>
77  </dependency>
78
79    </dependencies>
80
81    <build>
82      <plugins>
83        <plugin>
84          <groupId>org.springframework.boot</groupId>
85          <artifactId>spring-boot-maven-plugin</artifactId>
86          <configuration>
87            <excludes>
88              <exclude>
89                <groupId>org.projectlombok</groupId>
90                <artifactId>lombok</artifactId>
91              </exclude>
92            </excludes>
93          </configuration>
94        </plugin>
95      </plugins>
96    </build>
97
98  </project>
```

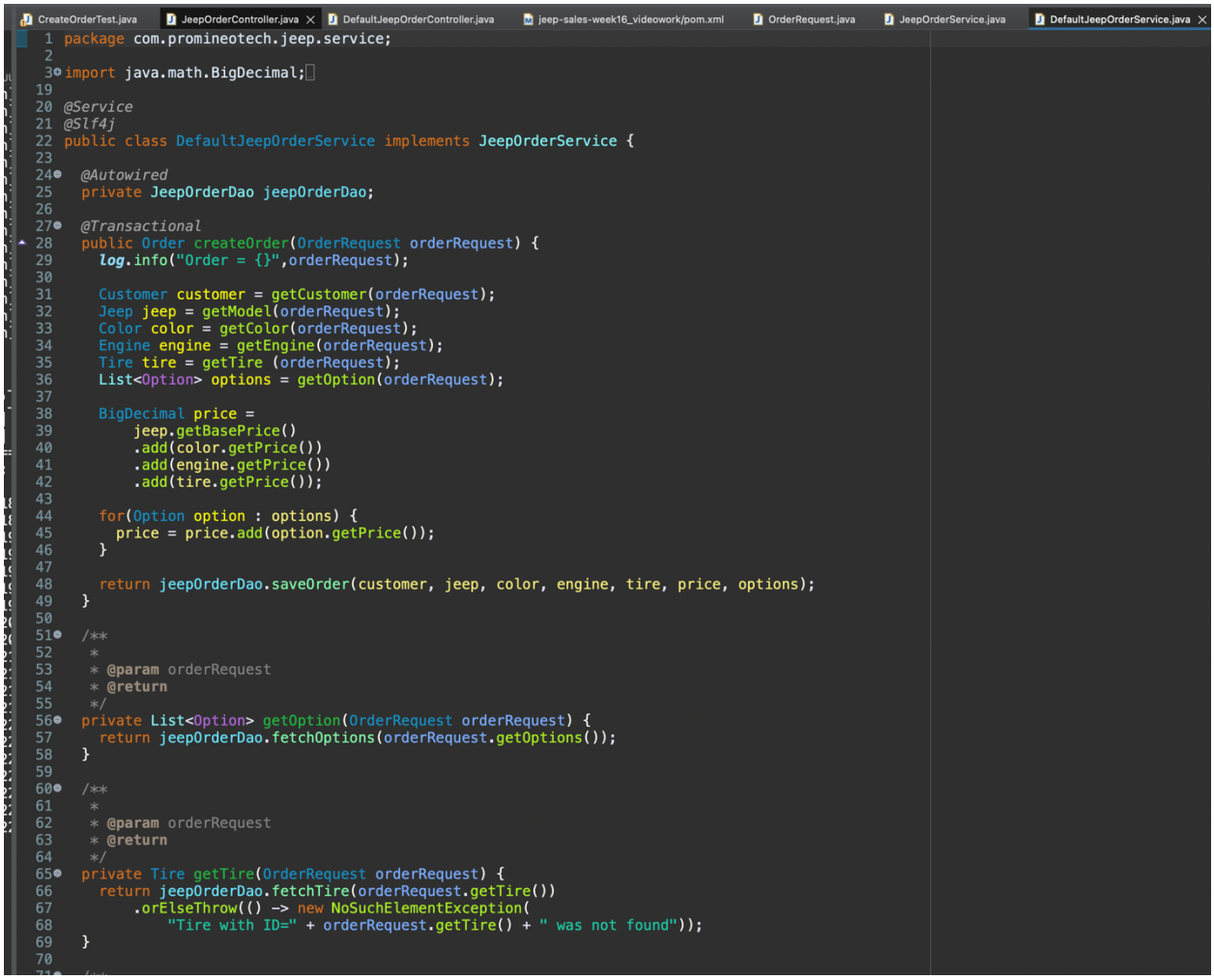
Overview Dependencies Dependency Hierarchy Effective POM pom.xml

CreateOrderTest.java | JeepOrderController.java | DefaultJeepOrderController.java | jeep-sales-week16_videowork/pom.xml | OrderRequest.java

```java
package com.promineotech.jeep.entity;

import java.util.List;

@Data
public class OrderRequest {
    @NotNull
    @Length(max = 30)
    @Pattern(regexp = "[A-Z_]")
  private String customer;

    @NotNull
  private JeepModel model;

    @NotNull
    @Length(max = 30)
    @Pattern(regexp = "[A-Z_]")
  private String trim;

    @Positive
    @Min(2)
    @Max(4)
  private int doors;

    @NotNull
    @Length(max = 30)
    @Pattern(regexp = "[A-Z_]")
  private String color;

    @NotNull
    @Length(max = 30)
    @Pattern(regexp = "[A-Z_]")
  private String engine;

    @NotNull
    @Length(max = 30)
    @Pattern(regexp = "[A-Z_]")
  private String tire;

  private List<String> options;
}

```

CreateOrderTest.java | JeepOrderController.java | DefaultJeepOrderController.java | jeep-sales-week16_videowork/pom.xml | OrderRequest.java | JeepOrderService.java

```java
package com.promineotech.jeep.service;

import com.promineotech.jeep.entity.Order;

public interface JeepOrderService {

  Order createOrder(OrderRequest orderRequest);

}
```


CreateOrderTest.java
JeepOrderController.java
DefaultJeepOrderController.java
jeep-sales-week16_videowork/pom.xml
OrderRequest.java
JeepOrderService.java
DefaultJeepOrderService.java
package com.promineotech.jeep.service;
import java.math.BigDecimal;
@Service
@Slf4j
public class DefaultJeepOrderService implements JeepOrderService {
@Autowired
private JeepOrderDao jeepOrderDao;
@Transactional
public Order createOrder(OrderRequest orderRequest) {
log.info("Order = {}",orderRequest);
Customer customer = getCustomer(orderRequest);
Jeep jeep = getModel(orderRequest);
Color color = getColor(orderRequest);
Engine engine = getEngine(orderRequest);
Tire tire = getTire (orderRequest);
List<Option> options = getOption(orderRequest);
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);
}
/**
*
* @param orderRequest
* @return
*/
private List<Option> getOption(OrderRequest orderRequest) {
return jeepOrderDao.fetchOptions(orderRequest.getOptions());
}
/**
*
* @param orderRequest
* @return
*/
private Tire getTire(OrderRequest orderRequest) {
return jeepOrderDao.fetchTire(orderRequest.getTire())
.orElseThrow(() -> new NoSuchElementException(
"Tire with ID=" + orderRequest.getTire() + " was not found"));
}

```
71    /**
72     *
73     * @param orderRequest
74     * @return
75     */
76    private Engine getEngine(OrderRequest orderRequest) {
77      return jeepOrderDao.fetchEngine(orderRequest.getEngine())
78          .orElseThrow(() -> new NoSuchElementException(
79              "Engine with ID=" + orderRequest.getEngine() + " was not found"));
80    }
81
82    /**
83     *
84     * @param orderRequest
85     * @return
86     */
87    private Color getColor(OrderRequest orderRequest) {
88      return jeepOrderDao.fetchColor(orderRequest.getColor())
89          .orElseThrow(() -> new NoSuchElementException(
90              "Color with ID=" + orderRequest.getColor() + " was not found"));
91    }
92
93    /**
94     *
95     * @param orderRequest
96     * @return
97     */
98    private Jeep getModel(OrderRequest orderRequest) {
99      return jeepOrderDao
100         .fetchModel(orderRequest.getModel(), orderRequest.getTrim(),
101             orderRequest.getDoors())
102         .orElseThrow(() -> new NoSuchElementException("Model with ID="
103             + orderRequest.getModel() + ", trim=" + orderRequest.getTrim()
104             + orderRequest.getDoors() + " was not found"));
105   }
106
107   /**
108    *
109    * @param orderRequest
110    * @return
111    */
112   private Customer getCustomer(OrderRequest orderRequest) {
113     return jeepOrderDao.fetchCustomer(orderRequest.getCustomer())
114         .orElseThrow(() -> new NoSuchElementException("Customer with ID="
115             + orderRequest.getCustomer() + " was not found"));
116   }
117 }
```

CreateOrderTest.java | JeepOrderController.java | *JeepOrderDao.java | DefaultJeepOrderDao.java

```java
package com.promineotech.jeep.dao;

import java.math.BigDecimal;

public interface JeepOrderDao {
  List<Option> fetchOptions(List<String> optionIds);
  Optional<Customer> fetchCustomer(String customerId);
  Optional<Jeep> fetchModel(JeepModel model, String trim, int doors);
  Optional<Color> fetchColor(String colorId);
  Optional<Engine> fetchEngine(String engineId);
  Optional<Tire> fetchTire(String tireId);

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

}
```

CreateOrderTest.java | JeepOrderController.java | *JeepOrderDao.java | DefaultJeepOrderDao.java

```java
package com.promineotech.jeep.dao;

import java.math.BigDecimal;

@Component
public class DefaultJeepOrderDao implements JeepOrderDao {
      @Autowired
      private NamedParameterJdbcTemplate jdbcTemplate;

      @Override
      public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire,
        BigDecimal price, List<Option> options) {

        SqlParams params = generateInsertSql(customer, jeep, color, engine, tire, price);

        KeyHolder keyHolder = new GeneratedKeyHolder();
        jdbcTemplate.update(params.sql, params.source, keyHolder);

        Long orderPK = keyHolder.getKey().longValue();
        saveOptions(options, orderPK);

        return Order.builder()
            .orderPK(orderPK)
            .customer(customer)
            .model(jeep)
            .color(color)
            .engine(engine)
            .tire(tire)
            .options(options)
            .price(price)
            .build();
      }

      private void saveOptions(List<Option> options, Long orderPK) {
        for(Option option : options) {
          SqlParams params = generateInsertSql(option, orderPK);
          jdbcTemplate.update(params.sql, params.source);
        }

      }
      /**
       *
       * @param option
       * @param orderPK
       * @return
       */
      private SqlParams generateInsertSql(Option option, Long orderPK) {
        SqlParams params = new SqlParams();

        // @formatter:off
        params.sql = ""
            + "INSERT INTO order_options ("
            + "option_fk, order_fk"
            + ") VALUES ("
            + ":option_fk, :order_fk"
            + ")";
        // @formatter:on

```

```
New  eateOrderTest.java | JeepOrderController.java | *JeepOrderDao.java | DefaultJeepOrderDao.java ×

86          params.source.addValue("option_fk", option.getOptionPK());
87          params.source.addValue("order_fk", orderPK);
88
89          return params;
90        }
91
92        /**
93         *
94         * @param customer
95         * @param jeep
96         * @param color
97         * @param engine
98         * @param tire
99         * @param price
100        * @return
101        */
102       private SqlParams generateInsertSql(Customer customer, Jeep jeep, Color color,
103           Engine engine, Tire tire, BigDecimal price) {
104         // @formatter:off
105         String sql = ""
106             + "INSERT INTO orders ("
107             + "customer_fk, color_fk, engine_fk, tire_fk, model_fk, price"
108             + ") VALUES ("
109             + ":customer_fk, :color_fk, :engine_fk, :tire_fk, :model_fk, :price"
110             + ")";
111         // @formatter:on
112
113         SqlParams params = new SqlParams();
114
115         params.sql = sql;
116         params.source.addValue("customer_fk", customer.getCustomerPK());
117         params.source.addValue("color_fk", color.getColorPK());
118         params.source.addValue("engine_fk", engine.getEnginePK());
119         params.source.addValue("tire_fk", tire.getTirePK());
120         params.source.addValue("model_fk", jeep.getModelPK());
121         params.source.addValue("price", price);
122
123         return params;
124       }
125
126       /**
127        *
128        */
129       @Override
130       public List<Option> fetchOptions(List<String> optionIds) {
131         if (optionIds.isEmpty()) {
132           return new LinkedList<>();
133         }
134
135         Map<String, Object> params = new HashMap<>();
136
137         // @formatter:off
138         String sql = ""
139             + "SELECT * "
140             + "FROM options "
141             + "WHERE option_id IN(";
142         // @formatter:on
143
```

CreateOrderTest.java | JeepOrderController.java | *JeepOrderDao.java | DefaultJeepOrderDao.java

```
142         // @formatter:on
143
144         for (int index = 0; index < optionIds.size(); index++) {
145           String key = "option_" + index;
146           sql += ":" + key + ", ";
147           params.put(key, optionIds.get(index));
148         }
149
150         sql = sql.substring(0, sql.length() - 2);
151         sql += ")";
152
153         return jdbcTemplate.query(sql, params, new RowMapper<Option>() {
154           @Override
155           public Option mapRow(ResultSet rs, int rowNum) throws SQLException {
156             // @formatter:off
157             return Option.builder()
158                 .category(OptionType.valueOf(rs.getString("category")))
159                 .manufacturer(rs.getString("manufacturer"))
160                 .name(rs.getString("name"))
161                 .optionId(rs.getString("option_id"))
162                 .optionPK(rs.getLong("option_pk"))
163                 .price(rs.getBigDecimal("price"))
164                 .build();
165             // @formatter:on
166           }
167         });
168       }
169
170       /**
171        *
172        */
173       @Override
174       public Optional<Customer> fetchCustomer(String customerId) {
175         // @formatter:off
176         String sql = ""
177             + "SELECT * "
178             + "FROM customers "
179             + "WHERE customer_id = :customer_id";
180         // @formatter:on
181
182         Map<String, Object> params = new HashMap<>();
183         params.put("customer_id", customerId);
184
185         return Optional.ofNullable(
186             jdbcTemplate.query(sql, params, new CustomerResultSetExtractor()));
187       }
188
189       /**
190        *
191        */
192       @Override
193       public Optional<Jeep> fetchModel(JeepModel model, String trim, int doors) {
194         // @formatter:off
195         String sql = ""
196             + "SELECT * "
197             + "FROM models "
198             + "WHERE model_id = :model_id "
199             + "AND trim_level = :trim_level "
200             + "AND num_doors = :num_doors";
201         // @formatter:on
202
```

CreateOrderTest.java | JeepOrderController.java | *JeepOrderDao.java | DefaultJeepOrderDao.java

```
201          // @formatter:on
202
203          Map<String, Object> params = new HashMap<>();
204          params.put("model_id", model.toString());
205          params.put("trim_level", trim);
206          params.put("num_doors", doors);
207
208          return Optional.ofNullable(
209              jdbcTemplate.query(sql, params, new ModelResultSetExtractor()));
210        }
211
212        /**
213         *
214         */
215        @Override
216        public Optional<Color> fetchColor(String colorId) {
217          // @formatter:off
218          String sql = ""
219              + "SELECT * "
220              + "FROM colors "
221              + "WHERE color_id = :color_id";
222          // @formatter:on
223
224          Map<String, Object> params = new HashMap<>();
225          params.put("color_id", colorId);
226
227          return Optional.ofNullable(
228              jdbcTemplate.query(sql, params, new ColorResultSetExtractor()));
229        }
230
231        /**
232         *
233         */
234        @Override
235        public Optional<Engine> fetchEngine(String engineId) {
236          // @formatter:off
237          String sql = ""
238              + "SELECT * "
239              + "FROM engines "
240              + "WHERE engine_id = :engine_id";
241          // @formatter:on
242
243          Map<String, Object> params = new HashMap<>();
244          params.put("engine_id", engineId);
245
246          return Optional.ofNullable(
247              jdbcTemplate.query(sql, params, new EngineResultSetExtractor()));
248        }
249
250        /**
251         *
252         */
253        @Override
254        public Optional<Tire> fetchTire(String tireId) {
255          // @formatter:off
256          String sql = ""
257              + "SELECT * "
258              + "FROM tires "
259              + "WHERE tire_id = :tire_id";
260          // @formatter:on
261
```

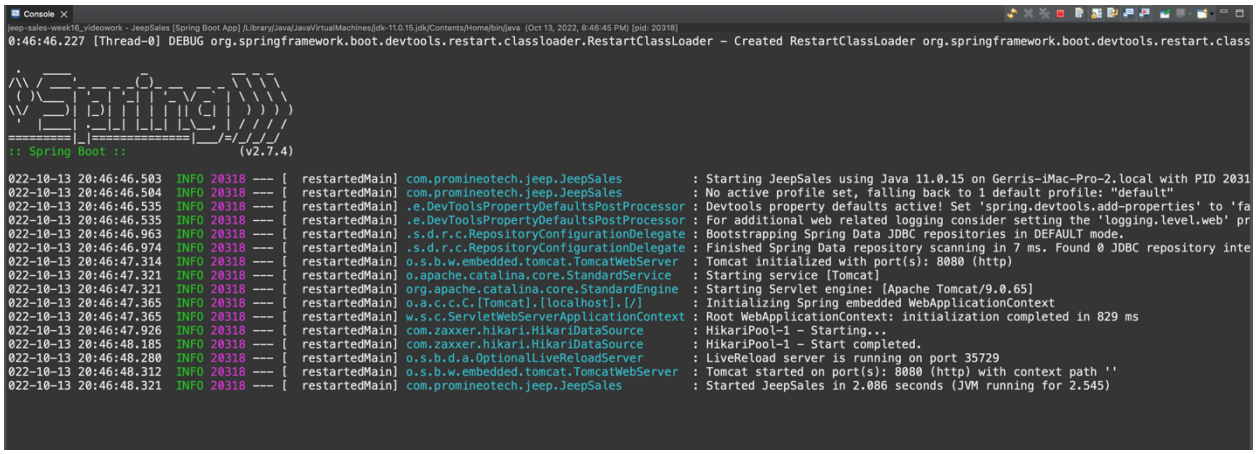
CreateOrderTest.java | JeepOrderController.java | *JeepOrderDao.java | DefaultJeepOrderDao.java

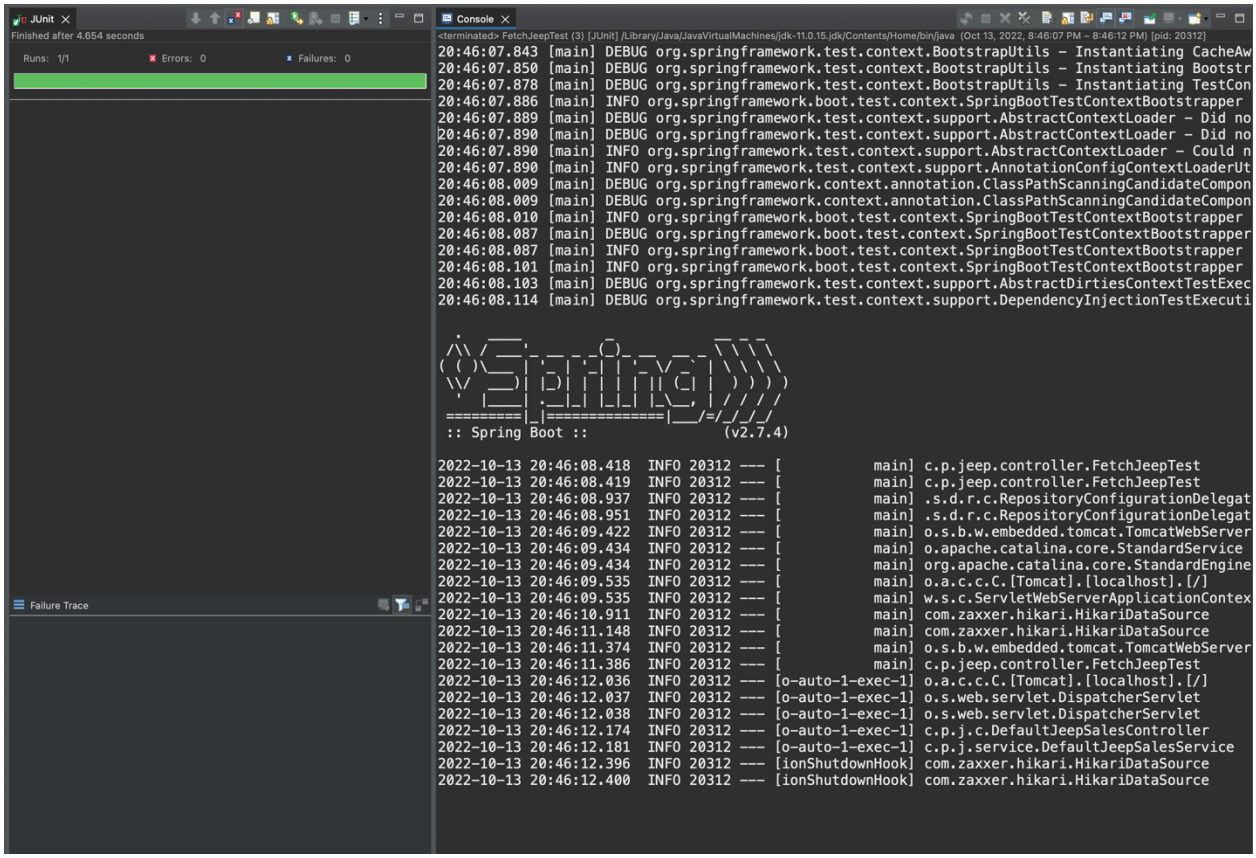
```
261
262         Map<String, Object> params = new HashMap<>();
263         params.put("tire_id", tireId);
264
265         return Optional.ofNullable(
266             jdbcTemplate.query(sql, params, new TireResultSetExtractor()));
267       }
268
269       /**
270        *
271        * @author Promineo
272        *
273        */
274       class TireResultSetExtractor implements ResultSetExtractor<Tire> {
275         @Override
276         public Tire extractData(ResultSet rs) throws SQLException {
277           rs.next();
278
279           // @formatter:off
280           return Tire.builder()
281               .manufacturer(rs.getString("manufacturer"))
282               .price(rs.getBigDecimal("price"))
283               .tireId(rs.getString("tire_id"))
284               .tirePK(rs.getLong("tire_pk"))
285               .tireSize(rs.getString("tire_size"))
286               .warrantyMiles(rs.getInt("warranty_miles"))
287               .build();
288           // @formatter:on
289         }
290       }
291
292       /**
293        *
294        * @author Promineo
295        *
296        */
297       class EngineResultSetExtractor implements ResultSetExtractor<Engine> {
298         @Override
299         public Engine extractData(ResultSet rs) throws SQLException {
300           rs.next();
301
302           // @formatter:off
303           return Engine.builder()
304               .description(rs.getString("description"))
305               .engineId(rs.getString("engine_id"))
306               .enginePK(rs.getLong("engine_pk"))
307               .fuelType(FuelType.valueOf(rs.getString("fuel_type")))
308               .hasStartStop(rs.getBoolean("has_start_stop"))
309               .mpgCity(rs.getFloat("mpg_city"))
310               .mpgHwy(rs.getFloat("mpg_hwy"))
311               .name(rs.getString("name"))
312               .price(rs.getBigDecimal("price"))
313               .sizeInLiters(rs.getFloat("size_in_liters"))
314               .build();
315           // @formatter:on
316         }
317       }
318
```


CreateOrderTest.java
JeepOrderController.java
*JeepOrderDao.java
DefaultJeepOrderDao.java
318
319 /**
320 *
321 * @author Promineo
322 *
323 */
324 class ColorResultSetExtractor implements ResultSetExtractor<Color> {
325 @Override
326 public Color extractData(ResultSet rs) throws SQLException {
327 rs.next();
328
329 // @formatter:off
330 return Color.builder()
331 .color(rs.getString("color"))
332 .colorId(rs.getString("color_id"))
333 .colorPK(rs.getLong("color_pk"))
334 .isExterior(rs.getBoolean("is_exterior"))
335 .price(rs.getBigDecimal("price"))
336 .build();
337 // @formatter:on
338 }
339 }
340
341 /**
342 *
343 * @author Promineo
344 *
345 */
346 class ModelResultSetExtractor implements ResultSetExtractor<Jeep> {
347 @Override
348 public Jeep extractData(ResultSet rs) throws SQLException {
349 rs.next();
350
351 // @formatter:off
352 return Jeep.builder()
353 .basePrice(rs.getBigDecimal("base_price"))
354 .modelId(JeepModel.valueOf(rs.getString("model_id")))
355 .modelPK(rs.getLong("model_pk"))
356 .numDoors(rs.getInt("num_doors"))
357 .trimLevel(rs.getString("trim_level"))
358 .wheelSize(rs.getInt("wheel_size"))
359 .build();
360 // @formatter:on
361 }
362 }
363
364 /**
365 *
366 * @author Promineo
367 *
368 */
369 class CustomerResultSetExtractor implements ResultSetExtractor<Customer> {
370 @Override
371 public Customer extractData(ResultSet rs) throws SQLException {
372 rs.next();
373

```
373
374             // @formatter:off
375             return Customer.builder()
376                 .customerId(rs.getString("customer_id"))
377                 .customerPK(rs.getLong("customer_pk"))
378                 .firstName(rs.getString("first_name"))
379                 .lastName(rs.getString("last_name"))
380                 .phone(rs.getString("phone"))
381                 .build();
382             // @formatter:on
383
384           }
385         }
386
387         class SqlParams {
388           String sql;
389           MapSqlParameterSource source = new MapSqlParameterSource();
390         }
391 }
```

## Screenshots of Running Application:






**URL to GitHub Repository:**

**https://github.com/geraldinedepaul17/BackEndClass/tree/main/Week%2016**