Simulation of the practical laboratory control of DP1

In this exercise, we will add the functionality to manage the products that will be sold in our pet clinic. To do this, we will perform a series of exercises based on features that we will implement in the system, and we will validate through unit tests. If you want to see the result of the tests, you can execute the command "*.\mvnw test*" in the root folder of the project. Each successfully passed test will be worth one point.

To start the control check you must accept the task of this practical control through the following link: https://classroom.github.com/a/UdwslXLm

By accepting this task, an individual unique repository will be created for you, you must use this repository to perform the control check and solve the exercises. You must submit the activity in EV associated to the control check providing as text the url address of your personal repository. Remember that you must also submit your solution to the control check.

The submission of your solution will be done by a single "*git push*" command to your individual repository. Remember to push before logging out of the computer and leaving the classroom, otherwise your attempt will be evaluated as unsubmitted.

Your first task in this control check will be to clone the repository (use the command "*git clone XXXX*" for this, where XXXX is the url of your repository). Next, you will need to import the project into your favorite development environment and start the exercises listed below. When importing the project, you will notice that the project has compilation errors. Don't worry, these errors will disappear as you implement the different exercises of the control check.

Important Note 1: Do not modify the class names or the signature (name, response type, and parameters) of the methods provided as source material for the control check. The tests used for evaluation depend on the classes and methods having the provided structure and names. If you modify them you will probably not be able to make the tests pass, and you will get a bad grade.

Important Note 2: Do not modify the unit tests provided as part of the project under any circumstances. Even if you modify the tests in your local copy of the project, they will be restored via a git command prior to running the tests for the final grade, so your modifications to the tests will not be taken into account.

Test 1 - Creating the Product Entity and its associated repository

Modify the "Product" class to be an entity. This entity will be hosted in the "org.springframework.samples.petclinic.product" package, and must have the following attributes and restrictions:

- An Integer type attribute named id that acts as a primary key in the relational database table associated to the entity.
- A "name" attribute of string type mandatory, with a minimum length of 3 characters and a maximum length of 50 characters.
- A "price" attribute of numeric type (double) mandatory, which can only take positive values (zero included).

Modify the "ProductRepository" interface hosted in the same package, so that it extends CrudRepository.

Test 2 - Creation of the Product Type Entity

Modify the class named "ProductType" hosted in the package "org.springframework.samples.petclinic.product" to be an Entity. This entity must have the following attributes and constraints:

- An Integer type attribute named "id" that acts as a primary key in the relational database table associated to the entity.
- A mandatory string attribute "name", with a minimum length of 3 characters and a maximum length of 50 characters. In addition, **the name of the product type must be unique.**

In addition, the method findAllProductTypes, from the product repository, must be annotated to execute a query which obtains all existing product types.

Test 3 - Modifying the database initialization script to include two products (and the associated product types)

Modify the database initialization script, so that the following products and product types are created:

Product 1:

- Id: 1
- Name: Wonderful dog necklace
- Price: 17.25

Product 2:

- Id: 2
- Name: Super Kitty Cookies
- Price: 50.0

Product type 1:

- Id: 1
- Name: Accessories

Type of product 2:

- Id: 2
- Name: Food

Test 4 - Create a one-way N to 1 relationship from Product to ProductType

In addition to creating a one-way N to 1 relationship from Product to ProductType, modify the database initialization script so that each product is associated with the corresponding product type.

Test 5 - Creation of a Product Management Service

Modify the ProductService class, so that it is a Spring business logic service, that allows to get all the products (as a list) using the repository. Do not modify the implementation of the other methods of the service for now.

Test 6 - Annotate the repository to get product types by name, and implement associated method in the product management service

Create a custom query that can be invoked through the product repository that gets a product type by name. Expose it through the product management service using the "getProductType(String name)" method.

Test 7- Creating a Product Type Formatter

Implement the formatter methods for the ProductType class (using the class called ProductTypeFormatter which should already be hosted in the same package we are working with). The formatter should display the products using their name string, and it should get a product type given its name (searching for it in the DB). Remember that if the formatter cannot parse an appropriate value from the provided text, it must throw a ParseException.

Test 8- Creating a custom query of products cheaper than a certain quantity

Create a custom query by annotating the "findByPriceLessThan" method in the repository, so that it takes a cost value as a parameter (parameter of type double) and returns all products cheaper than the given amount. Extend the product management service to include a method called "getProductsCheaperThan" that invokes the repository.

Test 9 - Creating a form for product creation/editing

Create a form to create a new product. This form must allow to specify the name, the price, and the type of product associated. The form must be available through the url:

http://localhost:8080/product/create

The form must be empty by default, and must contain two visible text inputs, a selector to choose the type of product, and a button to submit the data. The form data should be sent to the same address

using the post method. You may need to add some method to the product and product type management service.

The associated controller must be created as a method of the "ProductController" class in the same package, and it should pass to the form an object of type Product with the name "product" through the model.

The jsp page must be created inside the jsps folder (webapp/WEB-INF/jsp), in a folder named "products", and the file must be named "createOrUpdateProductForm.jsp".

Remember that as part of the implementation you must modify the application's security settings to allow only admin users (authority "admin") to access the form.

Test 10 - Creation of the Controller for the creation of new products.

Create a controller method in the previous class that responds to post type requests in the url and is responsible for validating the data of the new product, display the errors found if they exist through the form, and if there are no errors, store the product through the product management service. After saving the product, in case of success, it will show the home page of the application (welcome), without using redirection. Therefore, you must implement the "save" method of the product management service and invoke it from the controller.