Final controlcheck description

To start the control check, you must accept the task of this practical control through the following link: https://classroom.github.com/a/aS6-HMTh

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 by providing as text the URL address of your personal repository. Recall that you must also submit your solution to the control check.

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 in the document provided through the repository. When importing the project, it is possible that the project has compilation errors. Don't worry, these errors, if present, will disappear as you implement the exercises of the control check.

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 them through unit tests. If you want to see the results 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.

Note 1: Do not modify the class names or 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.

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 – Care entity

Modify the"Care"classhosted in the "org.springframework.samples.petclinic" package. care" to be an entity. This entity must have the following attributes and constraints:

- An integer attribute named "id" that acts as the primary key in the relational database table associated with the entity.
- A String attribute "name" that cannot be empty, that is unique (that is, there cannot be another Care with the same name in the database), and whose minimum length is 5 characters and a maximum of 30.
- A String attribute named "description". Empty descriptions are not allowed.
- A one-way N-to-N relationship with the PetType class that describes the type of pet to which the care can be applied.   The attribute that represents the relationship must be of type Set. This set cannot be empty, since any care should be applied to at least to one type of pet. You must also enable cascading of all JPA operations on this attribute.

Test 2 –CareProvision entity and its repository

Modify the "CareProvision" class to be an entity. This class is hosted in the package "org.springframework.samples.petclinic. care", and must have the following attributes and constraints:

- Integer attribute called "id" will act as the primary key in the relational database table associated with the entity.
- Create a one-way N-to-1 relationship from "CareProvision" to "Care" required, using as the attribute name in the "care" class. This relationship will indicate the care applied to the pet during the visit (haircut, flea removal, etc.).
- A double-precision floating-point numeric attribute called "duration", which indicates the duration in hours of service/care provision. It ismandatory, and only values greater than or equal to 0 are allowed.
- A one-way (**not** mandatory) N-1 relationship from "CareProvision" to "Visit" using an attribute called"visit". This relationship will indicate the visit during which the service/care is provided.

Modify the "CareProvisionRepository" interface hosted in the same package so that it extends to CrudRepository and it manages the cares applied to pets during visits (CareProvision).

In addition, the method "List<Care> findAllCares ()"   should be uncommented and annotated.

Test 3 – DB initialization script

Modify the database initialization script to create the following cares (Care) and care provisions (CareProvision):

Care 1:

- id: 1
- name: “Hair brushing”
- description: “We will brush the hair of your pet.”

Note: This care must be associatedwith dogs (PetType with id 2) and cats (PetType with id 1). You must insert the rows that are necessary in the intermediate table of the N to N relationship.

Care 2:

- id: 2
- name: “Chemical flea removal”
- description: “We will apply strong chemical products in the hair of your pet to remove any kind of flea or insect present.”

Note: This care must be associated with dogs (PetType with id 2) and cats (PetType with id 1). You must insert the rows that are necessary in the intermediate table of the N to N relationship.

CareProvision 1:

- id: 1
- visit_id: 1
- duratoin: 0.5
- care_id:2

CareProvision 2:

- id: 2
- visit_id: 2
- duration: 0.25
- care_id: 1

Test 4 – Create a N to N reflexive relationship on the Care entity

Create a one-way reflexive N-to-N relationship (with the Care entity itself), representing the set of cares incompatible with current care on the same visit. For example, if a brush and haircut session is performed, it is not advisable to perform a disinfectant hair treatment session with strong products on the same visit, since we can irritate the pet's skin. To do this, add an attribute called "incompatibleCares" of type Set<Care>. You must enable the execution of all JPA cascade operations for this attribute.

Modify the data initialization script to make Care 1 incompatible with Care 2 and vice versa.

Test 5 – Care management service

Modify the"CareService" class, so that it is a Spring business logic service, which allows to obtain all the cares provided (CareProvision). Do not modify the implementation of the other methods of the service by now.

Test 6 – Cares available for a pet type

Create a custom query that can be invoked through the care repository that obtains the cares available for a type of pet. To do this you must uncomment the method called "findCompatibleCares" and annotate it. That repository method should be used to implement the method that allows such care to be obtained in the care management service (the care service method called "getAllCompatibleCares").

Test 7– Care formatter

Implement the *formatter* methods for the Care class (which is already hosted in the same "care" package we are working with). The formatter must display the care as string using its name, and it must obtain a care given its name, looking for it in the database. In so doing, it must use of the care management service, using the method of the service that searches by name "getCare(String name)", not the repository. However, you may need to create or uncomment a method in the repository and invoke it from the service. Please, recall that if the formatter cannot get an appropriate value from the provided text, it must throw an exception of type "ParseException".

Test 8– Saving the cares provided to a pet during the visits

Implement the "saveProvidedCare" method of the care management service. If an incompatible care has already been during the visit, an exception of type "NonCompatibleCaresException" must be thrown (this class is already created in the cares package). In so doing, you must implement a custom query to find the cares performed during a visit in the care repository, and invoke it from the "getCaresPerformed(Integer visitId)" method of the care management service. In addition, the service must ensure that the care is compatible with the type of pet to which it provided (we must not allow to register haircut to a fish). If the type of pet that comes to the clinic during the visit is not among those compatible with the care, an exception of type "UnfeasibleCareException" should be thrown. In addition, in case of throwing any of those exceptions, the transaction associated with the operation must be rolled back .

Test 9 – ProvidedCare creation form

A form is provided in the project for the cares provided during visits. This form allows you to specify the care performed, and the duration of the application of such care. The jsp page of the form has been created inside the jsps folder (webapp/WEB-INF/jsp), in a subfolder named "cares", and the file is named "createOrUpdateProvidedCareForm.jsp".

Create a handler method that allows the form to be displayed. It must be available through the url:

http://localhost:8080/visit/<X>/care/create

Where <X> is the identifier of the visit to which the care will be associated.

The form must appear empty by default, and must contain a text field for the duration, and a selection field to set the care. The form data must be s to the same address using the post method. You may need to add some methods to the care management service.

The associated controller must be created as a method of the "CareController" class in the same package, and pass to the form an object of type ProvidedCare with name"providedCare" through the model. In addition, the collection of compatible cares for the pet visiting the clinic must be passed from the controller to the view (recall that the id of the Visit is provided through the URL). Such available cares will be provided under the name "cares". In order to get the visit you must use the pet management service (PetService has an appropriate method that has been created for this purpose). Remember that as part of the deployment you must modify the security settings of the application to allow administrators (authority "admin") use the form.

Test 10 – CareProvision save controller.

Create a controller method in the previous class that responds to post requests in the URL described above. Such controller method is responsible for validating the data of the care provided, showing the errors found if they exist through the form, and errors are not present, storing the new care provided through the care management service. Please note that, if the care to be stored is not compatible with the cares previously performed during that visit, or is not suitable for the type of pet of the visit, the corresponding exception must be captured and the error message "The selected care cannot be performed during this visit" should be visible in the field of care of the form. After saving the care provided, in case of success, redirection will be used to return to the home page of the application (“/”).