Control práctico de DP1 – Final

Enunciado

En este ejercicio, añadiremos un portfolio de servicios/cuidados adicionales que la clínica ofrecerá como complemento a las mascotas cuando vengan a las visitas al veterinario. Para ello realizaremos una serie de ejercicios basados en funcionalidades que implementaremos en el sistema, y validaremos mediante pruebas unitarias. Si desea ver el resultado que arrojarían las pruebas, puede ejecutarlas (bien mediante su entorno de desarrollo favorito, bien mediante el comando "*mvnw test*" en la carpeta raíz del proyecto). Cada prueba correctamente pasada valdrá un punto.

Para comenzar el control debe aceptar la tarea de este control práctico a través del siguiente enlace:

https://classroom.github.com/a/JvIKmsdr

Al aceptar dicha tarea, se creará un repositorio único individual para usted, debe usar dicho repositorio para realizar el control práctico. Debe entregar la actividad en EV asociada al control check proporcionando como texto la dirección url de su repositorio personal. Recuerde que además debe entregar su solución del control.

La entrega de su solución al control se realizará mediante un único comando "*git push*" a su repositorio individual. Recuerde que debe hacer push antes de cerrar sesión en la computadora y abandonar el aula, de lo contrario, su intento se evaluará como no presentado.

Su primera tarea en este control será clonar (use el comando "*git clone XXXX*" para ello, donde XXXX es la url de su repositorio). A continuación, deberá importar el proyecto en su entorno de desarrollo favorito y comenzar los ejercicios abajo listados. Al importar el proyecto, el mismo puede presentar errores de compilación. No se preocupe, si existen, dichos errores irán despareciendo conforme usted vaya implementando los distintos ejercicios del control.

Nota importante 1: No modifique los nombres de las clases ni la signatura (nombre, tipo de respuesta y parámetros) de los métodos proporcionados como material de base para el control. Las pruebas que se usan para la evaluación dependen de que las clases y los métodos tengan dicha la estructura y nombres proporcionados. Si los modifica probablemente no pueda hacer que pasen las pruebas, y obtendrá una mala calificación.

Nota importante 2: No modifique las pruebas unitarias proporcionadas como parte del proyecto bajo ningún concepto. Aunque modifique las pruebas en su copia local del Proyecto, éstas serán restituidas mediante un comando git previamente a la ejecución de las pruebas para la emisión de la nota final, por lo que sus modificaciones en las pruebas no serán tenidas en cuenta en ningún momento.

Nota importante 3: Mientras haya ejercicios no resueltos habrá tests que no funcionan y, por tanto, el comando "*mvnw install*" finalizará con error. Esto es normal debido a la forma en la que está planteado el control y no hay que preocuparse por ello. Si se quiere probar la aplicación se puede ejecutar de la forma habitual pese a que "mvn install" finalice con error.

Test 1 – Creación de le entidad Care

Modificar la clase "Care" alojada en el paquete "org.springframework.samples.petclinic.care" para que sea una entidad. Esta entidad debe tener los siguientes atributos y restricciones:

- Un atributo de tipo entero (Integer) llamado "id" que actúe como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- Un atributo de tipo cadena (String) llamado "name" que no puede ser vacío, de valor único (es decir, que no puede haber en la base de datos otro Care con el mismo nombre[1]), y cuya longitud mínima son 3 caracteres y la máxima 40.
- Un atributo de tipo entero (Integer) llamado "careDuration" que debe contener un valor entre 1 y 120. Este atributo representará la duración del tratamiento o cuidado aplicado a la mascota.
- Una relación N a N unidireccional con la clase PetType (tipo de mascota) que describa el tipo de mascota al que se puede aplicar aplicar ese tipo de cuidado. El atributo que representa la relación debe ser de tipo Set. Este conjunto no puede ser vacío, puesto que todo cuidado debe poder aplicarse al menos a un tipo de mascota. Además, debe habilitar la ejecución en cascada de todas las operaciones de JPA sobre este atributo.

Test 2 – Creación de la Entidad CareProvision y su repositorio asociado

Modificar la clase "CareProvision" para que sea una entidad. Esta clase está alojada en el paquete "org.springframework.samples.petclinic.care", y debe tener los siguientes atributos y restricciones:

- El atributo de tipo entero (Integer) llamado "id" actuará como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- Crear una relación de N a 1 unidireccional desde "CareProvision" hacia "Care" obligatoria, usando como nombre del atributo en la clase "care". Esta relación indicará el cuidado realizado a la mascota durante la visita (corte de pelo, desparasitación, etc.).
- El atributo de tipo cadena llamado "userRating", donde los owners nos dejarán un comentario que debe comenzar con una valoración de entre 0 y 9 estrellas. Por tanto, la cadena de la descripción debe comenzar con el texto "Care rated with X stars", donde X es un valor entero entre 0 y 9. La cadena puede contener cualquier otro texto libre que deseen los usuarios después de la valoración, para poder recibir sugerencias o quejas, por ejemplo, "Care rated with 9 stars, I am quite happy with the service!". La expresión regular que modela dicho comportamiento es `^Care rated with [0-9] stars.*$`
- Debe tener una relación N-1 unidireccional (**no** obligatoria) desde "CareProvision" hacia "Visit" utilizando un atributo llamado "visit". Esta relación indicará la visita durante la que se presta el servicio/cuidado/tratamiento.

Modificar el interfaz "CareProvisionRepository" alojado en el mismo paquete para que extienda a CrudRepository y permita gestionar cuidados realizados a mascotas durante las visitas (CareProvision).

Además, se debe descomentar y anotar el método "List<Care> findAllCares ()" en dicho repositorio para que permita obtener todos los cuidados existentes.

[1] Usar para ello la anotación @Column sobre la propiedad de la clase

Test 3 – Modificación del script de inicialización de la base de datos para incluir dos cuidados, asociados a perros y gatos, y dos prestaciones de cuidados en visitas

Modificar el script de inicialización de la base de datos, para que se creen los siguientes cuidados (Care) y prestaciones de cuidados (CareProvision):

Care 1:

- id: 1
- name: “Hair cut”
- careDuration: 30

Nota: Este cuidado debe estar asociado a perros (PetType con id 2) y hamsters (PetType con id 6), para ello deberá insertar las filas que sean necesarias en la tabla intermedia de la relación N a N. Tenga en cuenta que el orden en que se especifican las inserciones es importante para que la inicialización sea correcta.

Care 2:

- id: 2
- name: “Exotic shampoo cleaning”
- careDuration: 15

Nota: Este cuidado debe estar asociado a hamsters (PetType con id 6) y gatos (PetType con id 1), para ello deberá insertar las filas que sean necesarias en la tabla intermedia de la relación N a N. Tenga en cuenta que el orden en que se especifican las inserciones es importante para que la inicialización sea correcta.

CareProvision 1:

- id: 1
- visit_id: 1
- user_rating: Care rated with 8 stars
- care_id:1

CareProvision 2:

- id: 2
- visit_id: 2
- user_rating: Care rated with 9 stars, I am quite happy!
- care_id: 2

Test 4 – Creación de un Servicio de gestión de los cuidados para mascotas

Modificar la clase “CareService”, para que sea un servicio Spring de lógica de negocio, que permita obtener todos cuidados realizados (CareProvision). No modifique por ahora la implementación de los demás métodos del servicio.

Test 5– Creación de un Formatter de cuidados

Implementar los métodos del *formatter* para la clase Care (usando la clase llamada "CareFormatter" que está ya alojada en el mismo paquete "care" con el que estamos trabajando). El formatter debe mostrar los cuidados usando la cadena de su nombre, y debe obtener un cuidado dado su nombre, buscándolo en la BD (para esto debe hacer uso del servicio de gestión de cuidados, usando el método del servicio que busca por nombre "getCare(String nombre)", y no del repositorio, aunque quizás necesite crear o descomentar un método en el repositorio e invocarlo desde el servicio). Recuerde que, si el formatter no puede obtener un valor apropiado a partir del texto proporcionado, debe lanzar una excepción de tipo "ParseException".

Test 6 - Crear una relación reflexiva N a N en la entidad Care

Cree una relación N a N reflexiva (con la propia entidad Care) unidireccional, que represente el conjunto de cuidados incompatibles con el cuidado actual en la misma visita. Por ejemplo, si se realiza una sesión de cardado y corte de pelo, no es recomendable realizar una sesión de tratamiento capilar desinfectante con productos fuertes en la misma visita, puesto que podemos irritar el cuero cabelludo de la mascota. Pare ello, añada un atributo llamado "incompatibleCares" de tipo Set<Care>. Debe habilitar la ejecución de todas las operaciones de JPA en casada para este atributo.

Modifique el script de inicialización de los datos para hacer que el cuidado 1 sea incompatible con el cuidado 2 y viceversa.

Test 7 – Anotar el repositorio para obtener los cuidados (Care) disponibles para un tipo de mascota, que no sean incompatibles con otro tipo de cuidado dado

Crear una consulta personalizada que pueda invocarse a través del método "getAllCompatibleCares" del servicio de gestión anterior. La consulta personalizada debe obtener los cuidados disponibles para un tipo de mascota especificada mediante un parámetro, que no sean incompatibles con otro tipo de cuidado que se proporcionará también como parámetro (y que se supone que ha sido proporcionado ya a la mascota durante la visita).

Test 8– Registro de cuidados realizados a una mascota durante una visita.

Implementar el método "save" del servicio de gestión de cuidados. Si durante la visita ya se ha prestado un cuidado que es incompatible con el que se desea grabar, se debe lanzar una excepción de tipo "NonCompatibleCaresException" (esta clase está ya creada en el paquete cares). Para poder hacer esto deberá crear una consulta personalizada en el repositorio que obtenga los cuidados proporcionados durante una visita, y exponerla a través del método "getCaresProvidedInVisitById(Integer visitId)" del servicio de gestión de cuidados. Además, el servicio debe asegurarse de que el cuidado es compatible con el tipo de mascota al que se pretende prestar (no debemos permitir registrar cuidaos de corte de pelo a peces, por ejemplo). Si el tipo de la mascota que viene a la clínica durante la visita no está entre las compatibles con el cuidado, deberá lanzarse una excepción de tipo "UnfeasibleCareException". Además, en caso de lanzarse cualquiera de las dos excepciones, la transacción asociada a la operación de guardado del cuidado debe echarse atrás (hacer rollback).

Test 9 – Creación de método que permita obtener todos los cuidados proporcionados con paginación

Se pide implementar un método que permita devolver un listado de cuidados proporcionados paginado. El método se deberá implementar en el servicio de gestión de cuidados (se ha creado un método vacío llamado ”getPaginatedCareProvisions” para ello en dicha clase) y habrá que modificar el repositorio creando un método que devuelva los datos paginados, invocándolo desde el servicio.

Test 10 – Creación de Controlador para la creación de nuevos cuidados prestados durante las visitas.

Crear un método de controlador en la clase anterior que responda a peticiones tipo post en la url:

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

Donde <X> es el identificador de la visita al que se asociará el cuidado.

Este método debe encargarse de validar los datos del cuidado administrado, mostrar los errores encontrados si existieran a través del formulario proporcionado en el fichero “cares/createOrUpdateProvidedCareForm”, y si no existen errores, almacenar el nuevo cuidado prestado a través del servicio de gestión de cuidados prestados. Tras grabar el cuidado proporcionado a la mascota, en caso de éxito, se usará redirección para volver a la página de inicio de la aplicación (“/”).

En caso de que el servicio de gestión de cuidados lance alguna de las dos excepciones se debe mostrar la página de error personalizada “cares/InvalidCare.jsp”. Para ello debe hacerse uso del objeto de configuración de controladores proporcionado en la clase “ExceptionHandlingControllerAdvice”, y anotar el método de gestión de las excepciones.