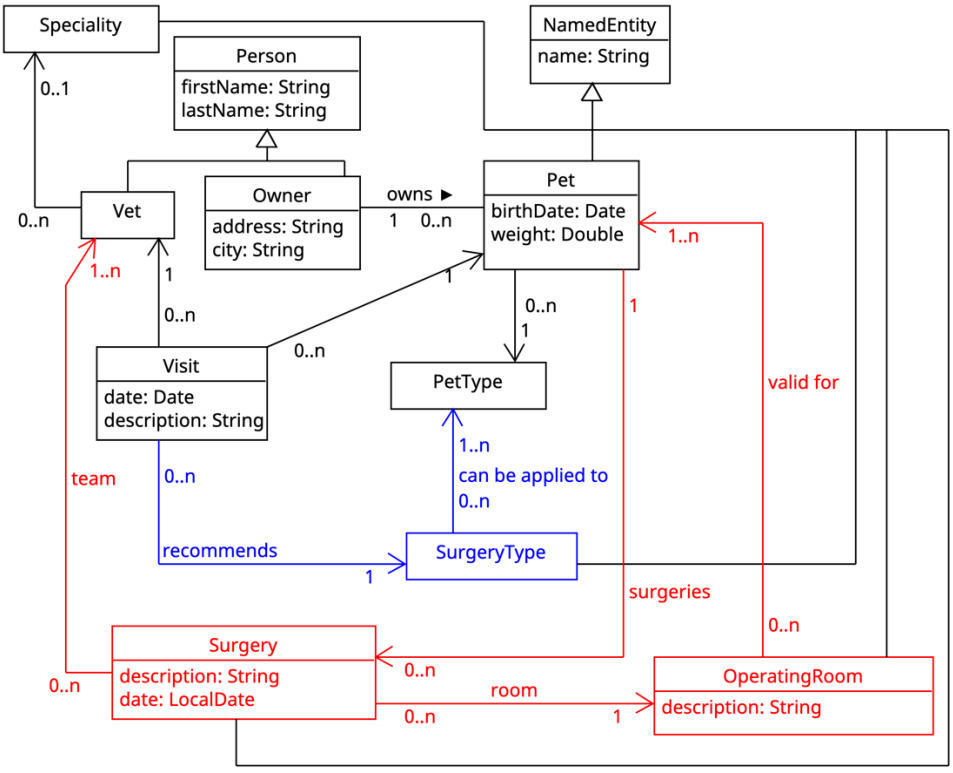
Control práctico de DP1 2023-2024 (Segundo control-check)

Enunciado

En este ejercicio, añadiremos la funcionalidad de gestión de cirugías a las mascotas. En concreto, tendremos la clase “Surgery”, que representa la realización de una cirugía a una mascota por un equipo de veterinarios, y la clase “OperatingRoom”, que representa las distintas salas de operaciones disponibles en la clínica para realizar la cirugía. Cada sala de operaciones está preparada para tratar a un conjunto específico de tipos de mascotas. Por ejemplo, podemos tener las salas 1 y 2 preparadas para perros y gatos y la sala 3 preparada para peces y aves. Además, tenemos también ya implementada la clase “SurgeryType”, que representa un tipo de cirugía concreto y se relaciona con aquellos PetType a los que ese tipo de cirugía se puede aplicar.

El diagrama UML que describe las clases y relaciones con las que vamos a trabajar es el siguiente:



Las clases para las que realizaremos el mapeo objeto-relacional como entidades JPA se han señalado en rojo. Las clases en azul son clases que se proporcionan ya mapeadas pero con las que se trabajará durante el control de laboratorio.

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 en backend, puede ejecutarlas (bien mediante su entorno de desarrollo favorito, bien mediante el comando “*mvnw test*” en la carpeta raíz del proyecto). Cada ejercicio correctamente resuelto valdrá un punto, el número de casos de prueba de cada ejercicio puede variar entre uno y otro y la nota se calculará en base al

porcentaje de casos de prueba que pasan. Por ejemplo, si pasan la mitad (50%) de los casos de prueba de un ejercicio, en lugar de un punto usted obtendrá un 0.5.

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

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

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 (recuerde que si va a usar los equipos del aula para realizar el control necesitará usar un token de autenticación de GitHub como clave, tiene un documento de ayuda a la configuración en el propio repositorio del control). 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 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 "mvnw install" finalice con error.

Nota importante 4: La descarga del material de la prueba usando git, y la entrega de su solución con git a través del repositorio GitHub creado a tal efecto forman parte de las competencias evaluadas durante el examen, por lo que no se aceptarán entregas que no hagan uso de este medio, y no se podrá solicitar ayuda a los profesores para realizar estas tareas.

Nota importante 5: No se aceptarán como soluciones válidas proyectos cuyo código fuente no compile correctamente o que provoquen fallos al arrancar la aplicación en la inicialización del contexto de Spring. Las soluciones cuyo código fuente no compile o incapaces de arrancar el contexto de Spring serán evaluadas con una nota de 0.

Test 1 – Creación de las entidades Surgery y OperatingRoom y sus repositorios asociados

Modificar las clases “Surgery” y “OperatingRoom” para que sean entidades. Estas clases están alojadas en el paquete “org.springframework.samples.petclinic.surgery”, y deben tener los siguientes atributos y restricciones:

Para ambas clases:

- 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.
- Un atributo de tipo cadena de caracteres (String) llamado “name” obligatorio (no puede ser nulo), que debe tener una longitud mínima de 3 caracteres y máxima de 50 y que no puede estar formada por caracteres vacíos (espacios, tabuladores, etc.).
- El atributo de tipo cadena caracteres (String) llamado “Description” opcional.

Para la clase Surgery:

- Un atributo de tipo fecha (LocalDate) llamado “date”, que representa la fecha en que se realiza la cirugía. Seguirá el formato “dd/MM/yyyy” (puede usar como ejemplo la clase Pet y su fecha de nacimiento para ver cómo se especificar dicho formato, pero nótese que el patrón del formato es distinto). Este atributo debe ser obligatorio y se almacenará en la BD con el nombre de columna “surgery_date”.

No modifique por ahora las anotaciones @Transient de las clases. Modificar las interfaces “SurgeryRepository” y “OperationRoomRepository” alojadas en el mismo paquete para que extiendan a CrudRepository. No olvide especificar sus parámetros de tipo.

Test 2 – Creación de relaciones entre las entidades

Elimine las anotaciones @Transient de los métodos y atributos que las tengan en las entidades creadas en el ejercicio anterior, así como la del atributo “surgeries” de la clase Pet. Se pide crear las siguientes relaciones entre las entidades:

- Una relación unidireccional desde “Pet” hacia “Surgery” que exprese la que aparece en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades, usando el atributo “surgeries” de la clase “Pet”.
- Una relación unidireccional desde “Surgery” hacia “Vet” que represente la que aparece en el diagrama UML respetando sus cardinalidades, usando el atributo “surgeryTeam”.
- Una relación unidireccional desde “OperatingRoom” hacia “PetType” que represente la que aparece en el diagrama UML respetando sus cardinalidades, usando el atributo “validFor”.
- Una relación unidireccional desde “Surgery” hacia “OperatingRoom” que exprese la que aparece en el diagrama UML respetando sus cardinalidades, usando el atributo “room” de la clase “Surgery”.

Test 3 – Modificación del script de inicialización de la base de datos para incluir dos cirugías y dos salas de operaciones

Modificar el script de inicialización de la base de datos, para que se creen las siguientes cirugías (Surgery) y salas de operaciones (OperationRoom):

Surgery 1:

- id: 1
- name: “Dental extraction”
- description: null
- date: 16/12/2023
- room: OperatingRoom con id=1

Surgery 2:

- id: 2
- name: “Bladder surgery”
- description: “Removal of a bladder stone.”
- date: 17/12/2023
- room: OperatingRoom con id=2

OperatingRoom 1:

- id: 1
- name: “Sala A”
- description: ”Sala de operaciones para perros y gatos”

OperatingRoom 2:

- id: 2
- name: “Sala B”
- description: ”Sala de operaciones para reptiles y tortugas”

Tenga en cuenta que el orden en que aparecen los INSERT en el script de inicialización de la base de datos es relevante al definir las asociaciones.

Test 4 – Modificación del script de inicialización de la base de datos para relacionar las cirugías y salas de operaciones con mascotas y tipos de mascota

Modificar este script de inicialización de la base de datos para que:

- La Pet cuyo id es 3 se asocie con la Surgery con id=1
- La Pet cuyo id es 1 se asocie con la Surgery con id=2
- La Surgery cuyo id es 1 se asocie con los Vet con ids 2 y 4.
- La Surgery cuyo id es 2 se asocie con los Vet con ids 1 y 5.
- La OperatingRoom con id=1 se asocie con los PetType con ids 1 y 2
- La OperatingRoom con id=2 se asocie con los PetType con ids 3, 4 y 7

Tenga en cuenta que el orden en que aparecen los INSERT en el script de inicialización de la base de datos es relevante al definir las asociaciones.

Test 5 – Creación de servicios de gestión de las cirugías y salas de operaciones

Modificar las clases "SurgeryService" y "OperatingRoomService", para que sean un servicio Spring de lógica de negocio y proporcionen una implementación a los métodos que permitan:

1. Obtener todas las cirugías/salas de operaciones (*Surgery / OperatingRoom*) almacenados en la base de datos como una lista usando el repositorio (método *getAll* en cada uno de los servicios correspondientes).
2. Grabar una cirugía/sala de operaciones (*Surgery / OperatingRoom*) en la base de datos (método *save*).

Todos estos métodos **deben ser transaccionales.** No modifique por ahora la implementación del resto de métodos de los servicios.

Test 6 – Anotar el repositorio de veterinarios con una consulta compleja

Modificar la consulta personalizada que puede invocarse a través del método "findMoreActiveVets" del repositorio de veterinarios "VetRepository" (alojado en el paquete org.springframework.samples.petclinic.vet) que recibe como parámetro dos fechas determinadas (que definen un rango de fechas donde la primera es anterior a la segunda), un conjunto de tipos de mascotas, y un entero que representa un umbral de número de visitas. El objetivo es que devuelva todos los veterinarios que han atendido a un número de visitas mayor o igual que el umbral del número de visitas durante el intervalo de tiempo definido en los parámetros para los tipos de mascotas especificados también por parámetro.

A continuación, se muestra un ejemplo. Sea la siguiente tabla de visitas y entre el 12 de noviembre de 2023 y el 11 de diciembre de 2023:

Pet	PetType[1]	Date (from Visit)	Vet
{id:1, name:"Leo"}	{id:, name:"Cat"}	13/11/2023	{id:4,last_name:"Ortega"}
{id: 3, name: "Rosy"}	{id: 2, name:"Dog"}	14/11/2023	{id:5,last_name:"Stevens"}
{id: 2, name:"Basil"}	{id: 6, name:"Hamster"}	17/11/2023	{id:5,last_name:"Stevens"}
{id: 4, name:"Jewel"}	{id: 2, name:"Dog"}	18/11/2023	{id:4,last_name:"Ortega"}
{id:10, name:"Mulligan"}	{id: 2, name:"Dog"}	29/11/2023	{id:1,last_name:"Carter"}
{id:12, name:"Lucky"}	{id: 2, name:"Dog"}	05/12/2023	{id:1,last_name:"Carter"}
{id:12, name:"Lucky"}	{id: 2, name:"Dog"}	10/12/2023	{id:1,last_name:"Carter"}

Si invocamos al método con los siguientes valores de los parámetros: petTypes={"dog","hamster"}, start=12 de noviembre de 2023, end=10 de diciembre de 2023, threshold=3. El resultado debería ser un conjunto que contenga al veterinario cuyo apellido es "Carter" puesto que para perros y hámsters en esas fechas atendió a 3 perros (y ningún hámster). Nótese que el veterinario de apellido "Stevens" no aparecería puesto que durante el periodo indicado solamente atendió a dos mascotas del tipo indicado (1 perro y 1 hámster).

[1] Obtained for the pet of the corresponding visit

Test 7 – Creación del controlador para devolver los tipos de cirugía disponibles

Crear un método en el controlador "*SurgeryTypeController"* (alojado en el paquete org.springframework.samples.petclinic.surgery) que permita devolver todos los tipos de cirugía existentes. El método debe responder a peticiones tipo GET en la URL:

http://localhost:8080/api/v1/surgerytypes

Así mismo debe crear un método para devolver un tipo de cirugía concreto, este método debe responder a peticiones en la url http://localhost:8080/api/v1/surgerytypes/X donde X es la Id del tipo de cirugía a obtener, y devolverá los datos del tipo de cirugía correspondiente. En caso de que no exista un tipo de cirugía con el id especificado el sistema debe devolver el código de estado 404 (NOT_FOUND).

Estos endpoint de la API asociados a la gestión de tipos de cirugía deberían estar accesibles únicamente para usuarios de tipo Vet (que tengan la authority "VET").

Test 8 – Modificar el servicio de gestión de Visitas para que no permita guardar recomendaciones de cirugía imposibles

Modificar el método save del servicio de gestión de visitas (VisitService) de manera que se lance la excepción (UnfeasibleSurgeryException) en caso de que se intente guardar una visita asociada a una recomendación de un tipo de cirugía que no aplica a las mascotas del tipo de la que ha venido a la visita. Por ejemplo, imaginen una visita donde el dueño trae a su pájaro y el veterinario le recomienda la cirugía de extracción de una pieza dental (que no puede ser realizada sobre ese tipo de mascotas). El método debe ser transaccional y hacer rollback de la transacción en caso de que se lance dicha excepción.

Test 9 – Implementar una prueba para un algoritmo de concesión de premios a veterinarios

En la clínica se ha decidido premiar a los veterinarios que han atendido más visitas. Para seleccionarlos se ha creado un método *Set<Vet> selectAwardedVets(Set<Visit> visits)* que recibe un conjunto de visitas y devuelve el conjunto de veterinarios que ha atendido más visitas de ese conjunto de visitas. Si el conjunto de visitas es vacío, el método debe devolver también una lista vacía. Este método se encuentra en la interfaz "AwardAlgorithm" que se encuentra en el paquete "org.springframework.samples.petclinic.disease".

Modifique la clase de pruebas llamada "AwardTest", que se encuentra en la carpeta "src/test/java/org/springframework/samples/petclinic/award/AwardTest.java" y especifique tantos métodos con casos de prueba como considere necesarios para validar el correcto funcionamiento del método. Nótese que la clase tiene un atributo de tipo AwardAlgorithm llamado algorithm. Debe usar dicho atributo como sujeto bajo prueba en todos sus métodos de prueba.

Para evaluar la calidad de las pruebas que ha implementado se utilizará en la evaluación distintas implementaciones alternativas de dicho método: algunas correctas y otras inválidas. Todas las pruebas que implemente deben pasar para las implementaciones correctas y, al menos una debe fallar para las implementaciones inválidas.

Su implementación del test **no debe usar mocks, ni anotaciones de pruebas de spring** (@DataJpaTest, @SpringBootTest,etc.), **ni tests parametrizados, y todos los métodos anotados con @Test deben ser sin parámetros**.

Test 10 – Creación de un componente React de listado de tipos de cirugía

Modificar el componente React proporcionado en el fichero "frontend/src/surgerytype/index.js" para que muestre un listado de las enfermedades disponibles en el sistema. Para ello debe hacer uso de la API lanzando una petición tipo GET contra la URL api/v1/surgerytypes.

Este componente debe mostrar los tipos de cirugía en una tabla (se recomienda usar el componente Table de reactstrap) incluyendo columnas para su nombre, y el conjunto de tipos de mascotas al que aplica este tipo de cirugía. Para esto último, el componente debe mostrar en la celda asociada una lista (preferiblemente no ordenada <ul>) con el nombre de los tipos de mascotas a los que aplica en la celda correspondiente.

Para poder lanzar este test y comprobar su resultado puede colocarse en la carpeta de frontend y ejecutar el comando npm test y pulsar 'a' en el menú de comandos de jest. Nótese que previamente debe haber lanzado al menos una vez el comando npm install para que todas las librerías de node estén instaladas