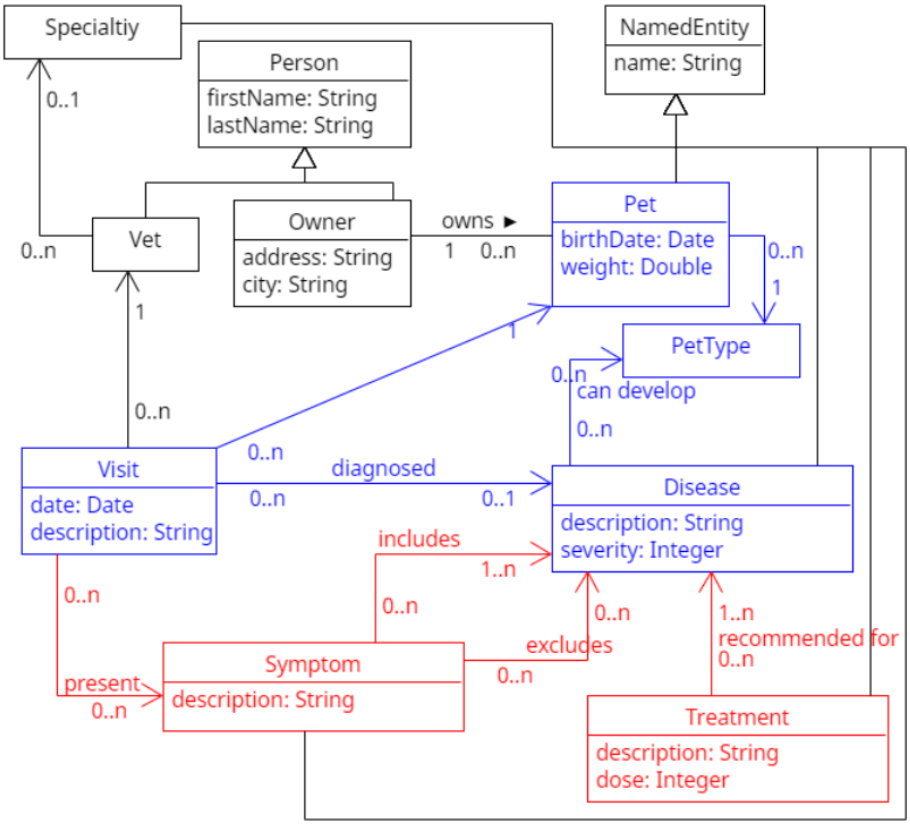
Control práctico de DP1 2023-2024 (Segundo control-check)

Enunciado

En este ejercicio, añadiremos la funcionalidad de gestión de enfermedades, síntomas y tratamientos médicos. Concretamente, se proporciona una clase "Disease" que representa a las enfermedades que pueden desarrollar las mascotas, que se relaciona con el tipo de mascotas que pueden sufrirlas. Además, tendremos las clases "Symptom" y "Treatment" que representan a los síntomas que pueden aparecer a consecuencia de una enfermedad y los tratamientos recomendados para cada enfermedad respectivamente. Además, se ha creado una relación que indica qué síntomas concretos excluyen que se trate de ciertas enfermedades, para ayudar a los veterinarios a realizar diagnósticos más precisos.

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/CSB9lvg4

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 Symptom y Treatment y sus repositorios asociados

Modificar las clases “Symptom” y “Treatment” para que sean entidades. Estas clases están alojadas en el paquete “org.springframework.samples.petclinic.disease”, 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 Symptom:

- El atributo de tipo entero (Integer) llamado “dose”, que representa el número de miligramos de tratamiento por kilogramo de peso del animal. Este atributo será obligatorio y tendrá un valor mínimo de 1.

No modifique por ahora las anotaciones @Transient de las clases. Modificar las interfaces “SymptomRepository” y “TreatmentRepository” 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 symptoms de la clase Visit. Se pide crear las siguientes relaciones entre las entidades. Cree una relación unidireccional desde “Visit” hacia “Symptom” que exprese la que aparece en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades, usando el atributo “symptoms” de la clase “Visit”. Este atributo debe ser obligatorio.

Además, se pide crear dos relaciones unidireccionales desde “Symptom” hacia “Disease” que representen las que aparecen en el diagrama UML, tenga en cuenta la cardinalidad que tienen (*recuerde que en este caso, al tratarse de una doble relación n a n entre las mismas entidades se trata de unas relaciones bastante exóticas)*, usando como nombre de los atributos “includedDiseases” y “excludedDiseases” en la clase “Symptom”. Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, algunos atributos pueden ser nulos puesto que la cardinalidad es 0..n pero otros no, porque su cardinalidad en el extremo navegable de la relación es 1..n.

Finalmente, se pide crear una relación unidireccional desde “Treatment“ hacia “Disease” que represente la que aparece en el diagrama. Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, el atributo no puede ser nulo y es obligatorio, puesto que la cardinalidad es 1..n en el extremo de Disease.

Test 3 – Modificación del script de inicialización de la base de datos para incluir dos síntomas y dos tratamientos

Modificar el script de inicialización de la base de datos, para que se creen los siguientes síntomas (Symptom) y tratamientos (Treatment):

Symptom 1:

- id: 1
- name: “Cough”
- description: null

Symptom 2:

- id: 2
- name: “Hair loss”
- description: “Hair loss in animals, also known as alopecia, can be a common and concerning symptom with various potential underlying causes.”

Treatment 1:

- id: 1
- name: “aspirin”
- description: ”Aspirin, also known by its generic name acetylsalicylic acid, is a widely used medication with analgesic (pain-relieving), antipyretic (fever-reducing), and anti-inflammatory properties.”
- dose: 12

Treatment 2:

- id: 2
- name: “paracetamol”
- description: ”Paracetamol, known as acetaminophen in the United States and Canada, is a widely used over-the-counter (OTC) medication with analgesic (pain-relieving) and antipyretic (fever-reducing) properties.”
- dose: 20

Test 4 – Modificación del script de inicialización de la base de datos para relacionar los tratamientos y los síntomas con las visitas y enfermedades

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

- El *Treatment* cuyo id es 1 se asocie con la *Disease* con id=2
- El Treatment cuyo id es 2 se asocie con la Disease con id=1
- El Symptom cuyo id es 1 tenga como posibles enfermedades incluidas la Disease con ids 2 y 3.
- El Symptom cuyo id es 2 tenga como posibles enfermedades incluidas la Diseases con ids 1 y 3.
- El Symptom cuyo id es 1 excluya como posible enfermedad la Disease con id=1.
- El Symptom cuyo id es 2 excluya como posible enfermedad la Disease con id=2.
- La Visit cuyo id es 1 tenga asociados los síntomas 1 y 2.

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 los síntomas y los tratamientos

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

1. Obtener todos los síntomas/tratamientos (*Symptom / Treatment*) almacenados en la base de datos como una lista usando el repositorio (método *getAll* en cada uno de los servicios correspondientes).
2. Grabar un síntoma/tratamiento (*Symptom / Treatment*) 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 Enfermedades con una consulta compleja

Modificar la consulta personalizada que puede invocarse a través del método "findEpidemicDiseases" del repositorio de tratamientos "DiseaseRepository" (alojado en el paquete org.springframework.samples.petclinic.disease) que reciba como parámetro un conjunto de tipos de mascotas, dos fechas determinadas (que definen un rango de fechas donde la primera es anterior a la segunda), y un entero que representa un número de diagnósticos. El objetivo es que devuelva todas las enfermedades diagnosticadas entre los tipos de mascotas especificados en alguna visita durante el periodo especificado, tales que estas enfermedades han sido diagnosticadas como mínimo el número de veces que indica el último parámetro. Este método permitirá encontrar enfermedades con visos de epidemia entre la población de una serie de tipos de mascotas durante un periodo indicado.

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

Pet	PetType[1]	Date (from Visit)	Diagnose
{id:1, name:"Leo"}	{id:, name:"Cat"}	13/11/2023	{id:1,name:"Rabies Shot"}
{id: 3, name: "Rosy"}	{id: 2, name:"Dog"}	14/11/2023	{id:1,name:"Rabies Shot"}
{id: 2, name:"Basil"}	{id: 6, name:"Hamster"}	17/11/2023	{id:1,name:"Rabies Shot"}
{id: 4, name:"Jewel"}	{id: 2, name:"Dog"}	18/11/2023	Null
{id:10, name:"Mulligan"}	{id: 2, name:"Dog"}	29/11/2023	{id:1,name:"Rabies Shot"}
{id:12, name:"Lucky"}	{id: 2, name:"Dog"}	05/12/2023	{id:2,name:"Flu"}
{id:12, name:"Lucky"}	{id: 2, name:"Dog"}	10/12/2023	{id:1,name:"Rabies Shot"}

Si invocamos al método con los siguientes valores de los parámetros: petTypes={"dog","hamster"}, startDate=12 de noviembre de 2023, endDate=10 de diciembre de 2023, diagnoses=2. El resultado debería ser un conjunto que contenga la enfermedad llamada "Rabies Shot" puesto que para perros y hámsters en esas fechas hubo cuatro visitas con diagnóstico, y 3 de ellas fueron de dicha enfermedad. Nótese que la enfermedad llamada "Flu" no aparecería puesto que durante el periodo indicado solamente se diagnosticó una vez en el periodo para el tipo de mascotas analizados.

[1] Obtained for the pet of the corresponding visit

Test 7 – Creación del controlador para devolver las Enfermedades disponibles

Crear un método en el controlador "*DiseaseController*" (alojada en el paquete org.springframework.samples.petclinic.disease) que permita devolver todas las enfermedades existentes. El método debe responder a peticiones tipo GET en la URL:

http://localhost:8080/api/v1/diseases

Así mismo debe crear un método para devolver una enfermedad concreta, este método debe responder a peticiones en la url http://localhost:8080/api/v1/diseases/X donde X es la Id de la enfermedad a obtener, y devolverá los datos de la enfermedad correspondiente. En caso de que no exista una enfermedad 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 enfermedades 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 diagnósticos imposibles

Modificar el método save del servicio de gestión de visitas (VisitService) de manera que se lance la excepción (UnfeasibleDiagnoseExcepction) en caso de que se intente guardar una visita asociada a una enfermedad que no pueden desarrollar las mascotas del tipo de la que ha venido a la visita. Por ejemplo, imaginen una visita donde el dueño trae a su serpiente y el veterinario lo diagnostica con juanetes en las patas traseras (que no puede ser desarrollada por 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 selección de enfermedades para el diseño de campañas de concienciación entre los dueños de mascotas

En la clínica se ha decidido comenzar a realizar campañas de concienciación entre los propietarios de mascotas para que traigan a sus mascotas en caso de detectar algunos síntomas y evitar así enfermedades graves. Para seleccionar las enfermedades (y los síntomas asociados) para los que se crearán las campañas se ha decidido crear un algoritmo de selección. Esencialmente dicho algoritmo se encargará de dado un conjunto de visitas con diagnósticos, devolver el conjunto de enfermedades para las que realizar las campañas. Para ello el algoritmo selecciona las enfermedades más graves diagnosticadas.

Además, hay algún caso límite a considerar, cuando la lista de enfermedades está vacía, el algoritmo debe devolver una lista también vacía.

La interfaz del algoritmo está especificada en la interfaz "CampaignDesignAlgorithm" que se encuentra en el paquete "org.springframework.samples.petclinic.disease". Y se proporcionan tres implementaciones del algoritmo (una correcta y dos incorrectas).

Modifique la clase de pruebas llamada "CampaignDesignTest" que se encuentra en la carpeta "scr/main/test/campaignDesign" y especifique tantos métodos con casos de prueba como considere necesarios para validar el correcto funcionamiento del algoritmo. Nótese que la clase tiene un atributo de tipo CampaignDesignAlgorithm llamado designAlgorithm, use dicho atributo como sujeto bajo prueba en todos sus métodos de prueba.

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 enfermedades

Modificar el componente React proporcionado en el fichero "frontend/src/disease/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/diseases.

Este componente debe mostrar las enfermedades en una tabla (se recomienda usar el componente Table de reactstrap) incluyendo columnas para su nombre, y el conjunto de tipos de mascotas que pueden desarrollar la enfermedad. 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 susceptibles de desarrollar la enfermedad 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.