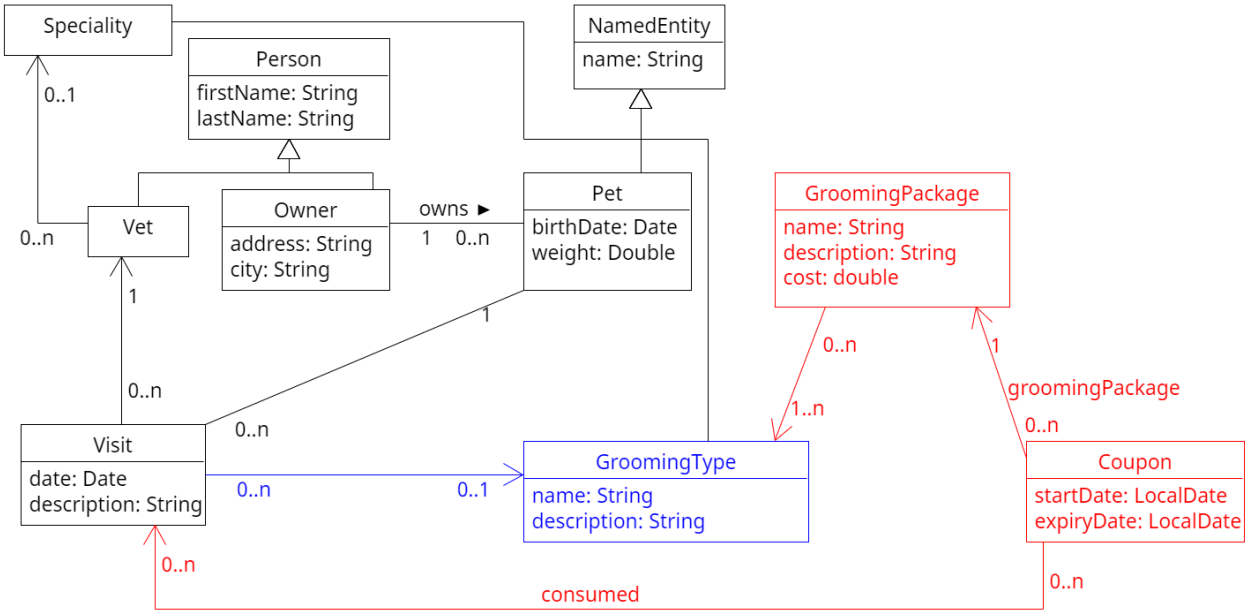
Control práctico de DP1 2023-2024 (Control check 1)

Enunciado

En este ejercicio, añadiremos la funcionalidad de ofrecer servicios de belleza a las mascotas (en inglés pet grooming) de forma combinada junto a las visitas a los veterinarios. Esto lo modelaremos con una clase llamada GroomingType que incluye una descripción. Ejemplos de GroomingType pueden ser “cepillado”, “baño”, “baño con masaje” o “corte de uñas”. Los GroomingType se agrupan en GroomingPackage, que incluyen un nombre, descripción y coste. Un GroomingPackage puede constar únicamente de un GroomingType como “baño con masaje” o de varios GroomingType como “cepillado + baño + corte de uñas”. Finalmente, los usuarios compran Coupon de un determinado GroomingPackage que van consumiendo en una o varias visitas. Los Coupon además tiene una fecha de validez determinada por los atributos startDate (inicio) y expirationDate (caducidad).

El diagrama UML que describe estas clases y relaciones 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, puedes 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/LVvKX4rB

Al aceptar dicha tarea, se creará un repositorio único individual para usted, debe usar dicho repositorio para realizar el control práctico.

La entrega de su solución al control se realizará (1) mediante un comando "*git push*" a su repositorio individual Y (2) entregando la actividad en EV asociada al control check proporcionando como texto la dirección url de su repositorio personal. Recuerde que debe hacer push antes de cerrar sesión en el ordenador 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 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.

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 la entidad GroomingPackage y su repositorio asociado

Modificar la clase "GroomingPackage" alojada en el paquete "org.springframework.samples.petclinic.grooming" 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 “description” obligatorio y no vacío que contiene la descripción del paquete.
- Un atributo de tipo real (double) llamado “cost” obligatorio que debe ser un número estrictamente positivo.

No modifique por ahora las anotaciones @Transient de la clase GroomingPackage. Modificar el interfaz “GroomingPackageRepository” alojado en el mismo paquete para que extienda a CrudRepository. No olvide especificar sus parámetros de tipo.

Test 2 – Creación de la entidad Coupon y su repositorio asociado

Modificar la clase “Coupon” para que sea una entidad. Esta clase está alojada en el paquete “org.springframework.samples.petclinic.grooming”, 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.
- Un atributo de tipo fecha (LocalDate) llamado “startDate”, que representa la fecha en que comienza la oferta, 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 “start”.
- El atributo de tipo fecha (LocalDate) llamado “expirationDate”, que representa la fecha en que termina la oferta, seguirá el formato “dd/MM/yyyy”. Este atributo debe ser obligatorio. En la base de datos se almacenará con el nombre de columna “finish”.

No modifique por ahora las anotaciones @Transient de la clase Coupon. Modificar el interfaz “CouponRepository” alojado en el mismo paquete para que extienda a CrudRepository. No olvide especificar sus parámetros de tipo.

Test 3 – Creación de relaciones entre las entidades

Elimine las anotaciones @Transient de los métodos y atributos que la tengan en las entidades creadas en los ejercicios 1 y 2. Se pide crear las siguientes relaciones entre las entidades.

1. Cree una relación unidireccional desde “GroomingPackage” hacia “GroomingType” que exprese la que aparece en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades, usando el atributo “contents” de la clase “GroomingType”. Este atributo debe ser obligatorio.
2. Crear una relación unidireccional desde “Coupon” hacia “GrromingPackage” que represente la que aparece en el diagrama UML, teniendo en cuenta la cardinalidad y usando el atributo “groomingPackage” en la clase “Coupon”
3. Crear una relación unidireccional desde “Coupon” hacia “Visit” que represente la que aparece en el diagrama UML, teniendo en cuenta la cardinalidad y usando el atributo “consumed” en la clase “Coupon”.

Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, el atributo groomingPackage no puede ser nulo puesto que la cardinalidad es 1.

Test 4 – Modificación del script de inicialización de la base de datos para incluir dos descuentos y dos ofertas

Modificar el script de inicialización de la base de datos, para que se creen las siguientes GroomingPackage y Coupon:

GroomingPackage 1:

- id: 1
- description: “Hair cut + Nails”
- cost: 50

GroomingPackage 2:

- id: 2
- description: “Bath + Legs massage”
- cost: 30

Coupon 1:

- id: 1
- startDate: 05-01-2023
- expiryDate: 21-02-2023
- grooming package: 1

Coupon 2:

- id: 2
- startDate: 20-12-2022
- expiryDate: 31-1-2023
- grooming package: 2

Test 5 – Modificación del script de inicialización de la base de datos para añadir información a las relaciones GroomingPackage – GroomingType y Coupon - Visit

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

- El *GroomingPackage* cuyo id es 1 se asocie con los *GroomingType* con id=1 e id=2
- El GroomingPackage cuyo id es 2 se asocie con los GroomingType con id=3 e id=5
- El Coupon cuyo id es 1 se consuma en la Visit con id=1
- El Coupon cuyo id es 2 se consuma en la Visit con id = 2 e id = 3

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.

Recuerda que en una asociación unidireccional N a M entre dos clases “A” y “B”, donde “bb” es el nombre del atributo de tipo “B” en la clase “A”, el nombre de la tabla en la base de datos correspondiente a la asociación se forma con “A_bb” y los nombres de las columnas de esa tabla son “a_id” y “bb_id”.

Test 6 – Creación de un Servicio de gestion de Grooming Packages y Coupons

Modificar las clases "CouponService" y "GroomingPackageService", para que seas un servicio Spring de lógica de negocio y proporcione una implementación a los métodos que permitan:

1. Obtener todos los *Coupon / GroomingPackages* almacenados en la base de datos como una lista usando el repositorio (método *getAll* en cada uno de los servicios correspondientes).
2. Grabar un *Coupon / GroomingPackage* 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 7 – Anotar el repositorio de Grooming Packages

Crear una consulta personalizada que pueda invocarse a través del repositorio de Grooming Packages "GroomingPackageRepository" (alojado en el paquete org.springframework.samples.petclinic.grooming) que reciba como parámetro dos fechas determinadas (start y end), que definen un rango de fechas donde la primera es anterior a la segunda, y un valor double (cost). El objetivo es que devuelva todos aquellos GroomingPackages de los cupones que tienen validez durante todo el período indicado por las fechas y cuyo coste sea inferior al establecido en el tercer parámetro. Para ello debe modificar la anotación del método llamado "findPackagesByDatesAndCost" especificando la consulta adecuada.

Test 8 – Creación del controlador para devolver los Grooming Types disponbiles

Crear un método en el controlador "*GroomingTypeController"* (alojada en el paquete org.springframework.samples.petclinic.grooming) que permita devolver todos los grooming types existentes. El método debe responder a peticiones tipo GET en la URL:

http://localhost:8080/api/v1/groomingtypes

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

Test 9 – Creación del controlador para crear nuevos grooming types

Crear un método de controlador en la clase anterior que responda a peticiones tipo post en la url http://localhost:8080/api/v1/groomingtypes y se encargue de validar los datos del grooming type, y devolver el código de estado 400 en caso de que el grooming type no sea válido. En caso de que no existan errores el controlador debe almacenar el grooming type a través del servicio de gestión de grooming types y devolver el código de estado 201 (CREATED) junto con el propio grooming type grabado en formato JSON en el cuerpo de la petición.

Test10 – Validación de fechas de visitas y cupones

Modificar el método addVisit del servicio de gestión de cupones, para que no permita relacionar una visita (Visit) con un cupón (Coupon) si la fecha de la visita no está en el intervalo de validez del cupón, es decir, entre la fecha de inicio del cupón y su fecha de expiración. En caso de que esto ocurra, se debe lanzar una excepción de tipo "*UnfeasibleCouponException*" (esta clase está ya creada en el paquete *grooming*). Además, en caso de lanzarse la excepción, la transacción asociada a la operación de guardado de la relación entre el cupón y la visita debe echarse atrás (hacer rollback).