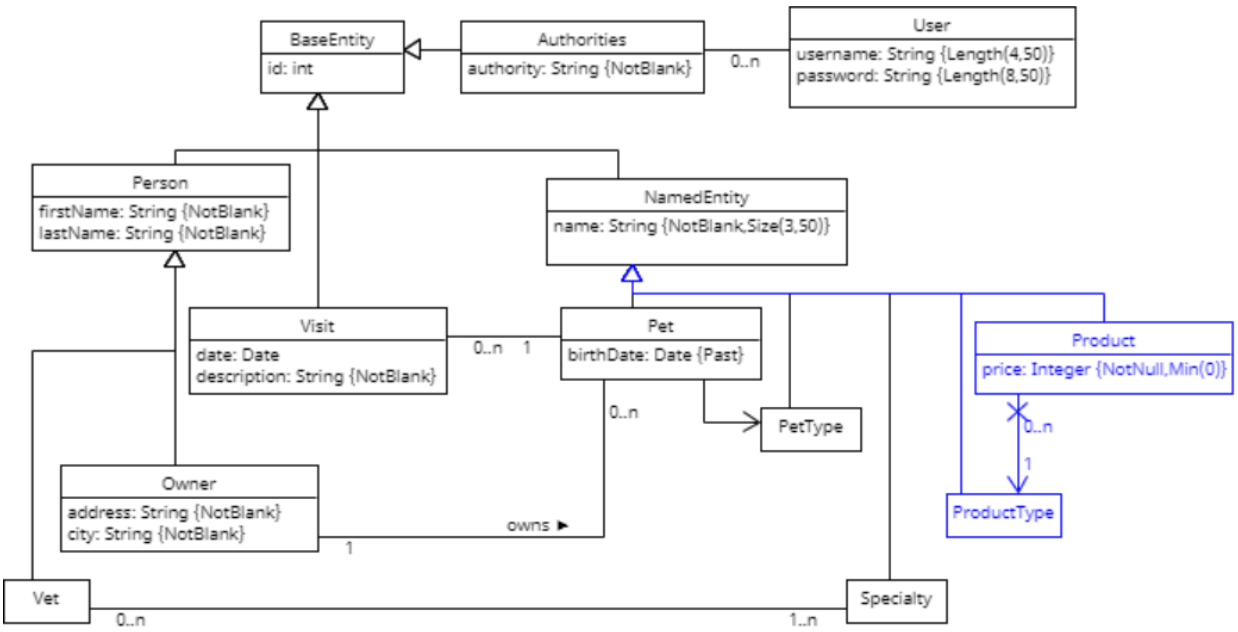
Simulación del control práctico de DP1

Enunciado

En este ejercicio, añadiremos la funcionalidad de gestión de productos que se venderán en nuestra clínica de mascotas. 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, puedes ejecutar el comando "*.\mvnw test*" en la carpeta raíz del proyecto. Cada prueba correctamente pasada valdrá un punto.



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/t-_ze9ld

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 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 Producto y su repositorio asociado

Se propone modificar la clase "Product" para que sea una entidad. Esta entidad estará alojada en el paquete "org.springframework.samples.petclinic.product", y debe tener los siguientes atributos y restricciones:

- Un atributo de tipo Integer llamado "id" que actúe como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- Un atributo "name" de tipo cadena obligatorio, y cuya longitud mínima son 3 caracteres y la máxima 50.
- Un atributo "price" de tipo numérico de entero (Integer) obligatorio, que solo podrá tomar valores positivos (cero incluido).
- Un atributo "type" de tipo ProductType obligatorio. No elimine la anotación @Transient por ahora.

Se propone modificar el interfaz “ProductRepository” alojado en el mismo paquete, para que extienda a CrudRepository.

Test 2 – Creación de la Entidad tipo de producto

Modificar la clase denominada “ProductType” alojarse en el paquete “org.springframework.samples.petclinic.product” para que sea una Entidad. Esta entidad debe tener los siguientes atributos y restricciones:

- Un atributo de tipo Integer llamado “Id” que actúe como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- Un atributo “name” de tipo cadena obligatorio, y cuya longitud mínima son 3 caracteres y la máxima 50. Además, **el nombre del tipo de producto debe ser único**.

Además, se debe anotar un método `findAllProductTypes`, del repositorio de productos, para que ejecute una consulta que permitan obtener todos los tipos de productos existentes como una lista.

Test 3 – Modificación del script de inicialización de la base de datos para incluir dos productos (y los tipos de productos asociados)

Modificar el script de inicialización de la base de datos, para que se creen los siguientes productos y tipos de productos:

Producto 1:

- Id: 1
- Name: Wonderful dog collar
- Price: 17

Producto 2:

- Id: 2
- Name: Super Kitty Cookies
- Price: 50

Tipo de producto 1:

- Id: 1
- Name: Accessories

Tipo de product 2:

- Id: 2
- Name: Food

Además, debe crear una relación de N a 1 unidireccional desde Product hacia ProductType, se propone modificar el script de inicialización de la base de datos para que cada producto quede asociado al tipo de producto correspondiente.

Test 4 – Creación de un Servicio de gestión de productos

Modificar la clase "ProductService", para que sea un servicio Spring de lógica de negocio, que permita obtener todos los productos (como una lista) y grabar los productos usando el repositorio. No modifique por ahora la implementación de los demás métodos del servicio. Recuerde que los métodos del servicio deberían estar anotados con @Transactional (y los parámetros de anotación adecuados en cada caso).

Test 5 – Anotar el repositorio para obtener los tipos de productos por nombre, e implementar método asociado en el servicio de gestión de productos

Crear una consulta personalizada que pueda invocarse a través del repositorio de productos que obtenga un tipo de producto por nombre. Exponerlo a través del servicio de gestión de productos mediante el método "`getProductType(String name)`". Recuerde que los métodos del servicio deberían estar anotados con @Transactional (y los parámetros de anotación adecuados en cada caso).

Test 6– Creación de consulta personalizada de productos más baratos que una cierta cantidad

Crear una consulta personalizada anotando un método llamado "`findByPriceLessThan`" en el repositorio, de manera que tome como parámetro un coste (parámetro de tipo Integer) y que devuelva todos los productos más baratos que la cantidad indicada. Extender el servicio de gestión de productos implementando el método llamado "`getProductsCheaperThan`" para que invoque al repositorio. Recuerde que los métodos del servicio deberían estar anotados con @Transactional (y los parámetros de anotación adecuados en cada caso).

Test 7 – Creación del Controlador y el método para la creación de nuevos productos.

Se propone crear un método de controlador en la clase "ProductController" que responda a peticiones tipo POST en la url "/api/v1/products" y se encargue de validar los datos del nuevo producto, mostrar los errores encontrados si existieran como parte de la respuesta (con código de estado 400 en ese caso), y si no existen errores, almacenar el producto a través del servicio de gestión de productos. Por tanto, deberá implementar el método "`save`" del servicio de gestión de productos. Recuerde que dicho método debe ser transaccional.

Test 8 – Obtención de producto por ID y modificación la representación de los productos a través de la API

Se propone modificar el controlador de productos para incluir una operación que permita obtener los datos de un producto en base a su id. El método debe responder a peticiones tipo GET en la url: "/api/v1/products/X", donde X es el id del producto.

Si se solicita un producto cuya id no existe en la base de datos se debe devolver una respuesta con código 404.

Se propone además modificar la representación de los datos proporcionados por la API para que el tipo de producto se muestre como una cadena de texto simple con su nombre. Ejemplo de JSON de representación de un producto:

{

“id”: 1,

“name”: “Wonderful dog collar”,

“price”: 17,

“type”: “Food”

}

Puede usar el mecanismo que considere oportuno para la codificación de los datos (crear un JSON encoder /serializador y anotar el atributo en la clase “Product” o usar un DTO en esta operación específica del controlador). Recuerde que para obtener el producto correspondiente debe usar el servicio de gestión de productos.

Test 9 – Modificación de productos existentes y configuración de seguridad

Se propone modificar el controlador de productos para incluir una operación que permita modificar un producto en base a su ID. El método debe responder a peticiones tipo PUT en la url: “/api/v1/products/X”, donde X es el id del producto. Para ello deberá crear también un deserializador o DTO que permita construir objetos con el tipo de producto expresado directamente como una cadena. Es importante que si el tipo de producto asociado al producto no está alojado en la base de datos, el valor que se establezca para el mismo en el producto generado por el deserializador (o la transformación del DTO al producto) sea nulo y por tanto se genere un error de tipo 400.

Si se solicita modificar un producto cuya id no existe en la base de datos se debe devolver una respuesta con código 404.

El controlador debe encargarse de validar los datos del producto modificado, mostrar los errores encontrados si existieran como parte de la respuesta (con código de estado 400 en ese caso), y si no existen errores, almacenar el producto a través del servicio de gestión de productos.

Recuerde configurar esta operación como accesible únicamente para los administradores del sistema (authority=”admin”).

Test 10 - Implementación de una regla de negocio que haga que no se puedan realizar subidas de precio en productos existentes superiores al 100% de su precio anterior

Se solicita implementar una regla de negocio en el sistema que impida realizar modificaciones en un producto que suponga un aumento de precio superior al 100% de su precio. Para ello debe hacer uso de la excepción “`UnfeaseibleProductModificationException`”, que deberá lanzarse por parte del servicio de gestión de productos en caso de que se detecte esta situación. El controlador deberá capturar dicha excepción y devolver un código de respuesta 400 en ese caso. La transacción asociada a la modificación del producto debe echarse atrás en caso de que se lance la excepción.