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.

Para comenzar el control debe aceptar la tarea de este control práctico a través del siguiente enlace: https://classroom.github.com/a/UdwslXLm

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 del 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, observará que el mismo presenta errores de compilación. No se preocupe, 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.

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 minima son 3 carácteres y la máxima 50.
- Un atributo "price" de tipo numérico de coma flotante (double) obligatorio, que solo podrá tomar valores positivos (cero incluido).

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 minima son 3 carácteres y la máxima 50. Además, **el nombre del tipo de producto debe ser único**.

¿Por qué aquí la rosi añade @Table (name = "product_type")?

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

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: 17.25

Se crean todos los productos en la base de datos. Primero los productType y luego los productos, ya que en el siguiente ejercicio dará error si se disponen al contrario.

Producto 2:

- Id: 2
- Name: Super Kitty Cookies
- Price: 50.0

La tabla product tiene otro atributo mas que es el tipo de producto si se deja en blanco o no se pone, da igual porque NO está puesto como @NotNull en la entidad.

Tipo de producto 1:

- Id: 1

- Name: Accessories

Tipo de product 2:

- Id: 2
- Name: Food

Test 4 - Crear una relación N a 1 unidireccional desde Product hacia ProductType

Además de 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 5 – Creación de un Servicio de gestion 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) usando el repositorio. No modifique por ahora la implementación de los demás métodos del servicio.

Test 6 – 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)”.

Test 7– Creación de un Formatter de tipos de producto

Implementar los métodos del formatter para la clase ProductType (usando la clase llamada ProductTypeFormatter que debe está ya alojada en el mismo paquete con el que estamos trabajando). El formatter debe mostrar los productos usando la cadena de su nombre, y debe obtener un tipo de producto dado su nombre buscándolo en la BD. Recuerde que si el formatter no puede parsear un valor apropiado a partir del texto proporcionado, debe lanzar una excepción de tipo ParseException.

Test 8– Creación de consulta personalizada de productos más baratos que una cierta cantidad

Crear una consulta personalizada anotando el método “findByPriceLessThan” en el repositorio, de manera que tome como parámetro un coste (parámetro de tipo double) y que devuelva todos los productos más baratos que la cantidad indicada. Extender el servicio de gestión de productos para que incluya a un método llamado “getProductsCheaperThan” que invoque al repositorio.

Test 9 – Creación de un formulario para la creación/edición de productos

Crear un formulario para crear un nuevo producto. Este formulario debe permitir especificar el nombre, el precio, y el tipo de producto asociado. El formulario debe quedar disponible a través de la url:

http://localhost:8080/product/create

El formulario debe aparecer por defecto vacío, y debe contener dos inputs de tipo texto visibles, un selector para escoger el tipo de producto, y un botón para enviar los datos. Los datos del formulario deben enviarse a la misma dirección usando el método post. Quizás necesite añadir algún método al servicio de gestión de productos y tipos de productos.

El controlador asociado debe crearse como método de la clase “ProductController” en el mismo paquete, y pasar al formulario un objeto de tipo Product con el nombre “product” a través del modelo. La página jsp debe crearse dentro de la carpeta de jsps (webapp/WEB-INF/jsp), en una carpeta llamada “products”, y el fichero debe llamarse “createOrUpdateProductForm.jsp”.

Recuerde que como parte de la implementación debe modificar la configuración de seguridad de la aplicación para permitir que solo los usuarios administradores (authority “admin”) accedan al formulario.

Test 10 – Creación del Controlador para la creación de nuevos productos.

Se propone crear un método de controlador en la clase anterior que responda a peticiones tipo post en la url y se encargue de validar los datos del nuevo producto, mostrar los errores encontrados si existieran a través del formulario, y si no existen errores, almacenar el producto a través del servicio de gestión de productos. Tras grabar el producto, en caso de éxito, se mostrará la página de inicio de la aplicación (welcome), sin usar redirección. Por tanto, deberá implementar el método “save” del servicio de gestión de productos.