# Control práctico de DP1 (C2)

## Enunciado

En este ejercicio, añadiremos la funcionalidad de gestión de guardias para los veterinarios de la 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 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/Jyvol_Vo**

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.

## Test 1 – Creación de la Entidad Watch y su repositorio asociado

Modificar la clase "Watch" para que sea una entidad. Esta clase está alojada en el paquete "org.springframework.samples.petclinic.watch", 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.
- El atributo de tipo fecha (LocalDate) llamado "date", que representa el día de la guardia, 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.
- El atributo de tipo hora (LocalTime) llamado "beginTime", que representa el momento en el que se empieza la guardia, seguirá el formato "HH:mm:ss" (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. En la base de datos se almacenará con el nombre de columna "start".
- El atributo de tipo hora (LocalTime) llamado "finishTime", que representa el momento en el que se termina la guardia, seguirá el formato "HH:mm:ss". Este atributo debe ser obligatorio. En la base de datos se almacenará con el nombre de columna "end".
- No elimine por ahora las anotaciones @Transient de los atributos Vet que representa al veterinario que realiza la guardia, y WatchType que representa el tipo de guardia (localizada, telefónica, de disponibilidad, etc.). No elimine tampoco las anotaciones @Transient de los métodos.

Modificar el interfaz "WatchRepository" alojado en el mismo paquete para que extienda a CrudRepository.

## Test 2 – Creación de la Entidad WatchType

Modificar la clase "WatchType" alojada en el paquete "org.springframework.samples.petclinic.watch" 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 "name" que no puede ser vacío y cuya longitud mínima son 5 caracteres y la máxima 30. No puede haber dos tipos de guardia con el mismo nombre (el valor debe ser único).
- Un atributo obligatorio de tipo boolean (Boolean) llamado "overtime" que representa si la guardia se realiza en horas extra.

Crear una relación de N a 1 unidireccional desde "Watch" hacia "WatchType", usando como nombre del atributo en la clase "type". Este atributo no puede ser nulo.

Crear una relación N a 1 unidireccional desde "Guardia" hacia "Vet" utilizando un atributo llamado "vet". Esta relación indicará el veterinario que realizará la guardia. Toda guardia debe estar asociado a un veterinario (no puede ser nulo).

Descomentar el método “List<WatchType> findAllWatchTypes()” en el repositorio de guardias (*WatchRepository*) y anotarlo para que ejecute una consulta que permita obtener todos los tipos de guardia existentes.

## Test 3 – Modificación del script de inicialización de la base de datos para incluir dos guardias y dos tipos de guardia

Modificar el script de inicialización de la base de datos, para que se creen las siguientes guardias (Watch) y tipos de guardia (WatchType):

Watch 1:

- id: 1
- date: 05-01-2022
- start: 07:00:00[1]
- finish: 15:00:00

Watch 2:

- id: 2
- date: 04-01-2022
- start: 16:30:00
- finish: 23:30:00

WatchType 1:

- id: 1
- name: Localised watch
- overtime: true

WatchType 2:

- id: 2
- name: Telephonic watch
- overtime: false

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

- La *Watch* cuyo id es 1 se asocie con el *Vet* con id=1 y *WatchType* con id=*2*
- La *Watch cuyo id es 2* se asocie con el *Vet* con id=2 y *WatchType* con id=*1*

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.

[1] Nótese que para insertar valores de tiempo en H2 puede usarse la siguiente sintaxis SQL: PARSEDATETIME('16:22', 'HH:mm')

## Test 4 – Creación de un Servicio de gestion de las guardias

Modificar la clase "WatchService", para que sea un servicio Spring de lógica de negocio y proporcione una implementación a los métodos que permitan:

1. Obtener todas las guardias (*Watch*) almacenados en la base de datos como una lista usando el repositorio (método *getAll*).
2. Obtener todo el conjunto de tipos de guardias (*WatchType*) registrados (método *getAllWatchTypes*).
3. Grabar una guardia (*Watch*) en la base de datos (método *save*).

Todos estos métodos **deben ser transaccionales.**

No modifique por ahora la implementación del resto de métodos del servicio.

## Test 5 – Anotar el repositorio de guardias para obtener las guardias de un veterinario que se solapan con un día y hora determinados

Crear una consulta personalizada que pueda invocarse a través del repositorio de guardias "WatchRepository" que reciba como parámetros un veterinario, así como una fecha y una hora determinadas. El objetivo es que devuelva todas aquellas guardias de ese veterinario en esa fecha y cuyas horas de inicio y fin engloban a la hora pasada por parámetro. Para ello debe descomentar el método llamado "findOverlappedWatches" y anotarlo. También debe descomentar el método "getOverlappedWatches" en el servicio de gestión de guardias "WatchService".

## Test 6 – Comprobación de la coherencia de las guardias.

Modificar el método "*save*" del servicio de gestión de guardias para que, si la guardia a realizar se solapa en el tiempo (en fecha y horas) con otra guardia del mismo veterinario, se lance una excepción de tipo "*UnfeasibleWatchException*" (esta clase está ya creada en el paquete *watch*). Además, en caso de lanzarse la excepción, la transacción asociada a la operación de guardado de la guardia debe echarse atrás (hacer rollback). **No utilice el método del repositorio creado en el test anterior**, en su lugar puede usar el método de la clase Watch llamado "`isConcurrentInTime`" para comprobar si dos guardias solapan en horas.

## Test 7 – Creación de un Formatter de tipos de guardia

Crear un método con una consulta personalizada en el repositorio de guardias para que obtenga un tipo de guardia por nombre.

Usar el método anterior para implementar el método "getWatchType(String name)" del servicio de gestión de guardias.

Implementar los métodos del *formatter* para la clase WatchType usando la clase llamada "WatchTypeFormatter" que está ya alojada en el mismo paquete "watch" con el que estamos trabajando. El *formatter* debe mostrar los tipos de guardias usando la cadena de su nombre y debe obtener un tipo de guardia dado su nombre buscándolo en la BD. Para esto debe hacer uso del método servicio de gestión

de guardias y no del repositorio. Recuerde que, si el formatter no puede obtener un valor apropiado a partir del texto proporcionado, debe lanzar una excepción de tipo "ParseException".

## Test 8 – Creación del controlador para mostrar un formulario de creación de guardias

Crear un método en el controlador "*WatchController"* que permita mostrar el formulario implementado en el fichero JSP llamado "*createOrUpdateWatchForm.jsp*" en la carpeta "*webapp/WEB-INF/jsp/watch*". Este formulario permite crear y editar guardias (*Watch*) especificando la fecha y las horas de inicio y fin. El formulario debe mostrarse en la URL:

http://localhost:8080/watch/create

El controlador debe pasar al formulario un objeto de tipo *Watch* con el nombre *watch* a través del modelo de la vista. Por defecto, el formulario debe aparecer vacío ya que estamos creando una nueva guardia. Los datos del formulario deben enviarse a la misma dirección usando el método post.

## Test 9 – Creación del controlador para la grabación de las nuevas guardias

Crear un método de controlador en la clase anterior que responda a peticiones tipo post en la url http://localhost:8080/watch/create y se encargue de validar los datos de la guardia, y mostrar los errores encontrados si existieran a través del formulario, y si no existen errores, almacenar la guardia a través del servicio de gestión de guardias.

Tras grabar la guardia, en caso de éxito, se mostrará la página de inicio de la aplicación (welcome) **usando redirección**. Recuerde que el formato de entrada de la fecha de la guardia debe ser "dd/MM/yyyy", y el formato de entrada de las horas debe ser "hh:mm:ss" en caso contrario las pruebas no pasarán (puede usar como ejemplo la clase Pet y su fecha de nacimiento para especificar los formatos).

En caso de que el servicio de gestión de guardias lance alguna la excepción "*UnfeasibleWatchException"* se debe mostrar la página de error personalizada "watch/InvalidWatch.jsp". Para ello debe hacer uso de la clase ExceptionHandlingControllerAdvice, y anotar el método `handleInvalidWatchException`.

## Test 10 – Creación de método que permita obtener todas las guardias realizadas con paginación

Se pide implementar un método que permita devolver un listado de guardias realizadas paginado. El método se deberá implementar en el servicio de gestión de guardias (se ha creado un método vacío llamado "getPaginatedWatches" para ello en dicha clase) y habrá que modificar el repositorio creando un método que devuelva los datos paginados, invocándolo desde el servicio.