Integrantes:

-Fernandez Mendez ; Carlos Alberto 1113693

-Esquivel Candela; LU :1134110

-Zarlenga, Ignacio ; LU 1132325

-Vidal, Manuel ; LU: 1146916

-Molina, Lucas; LU: 1146573

PATRONES

Mvc:

Relacionado a la lógica y reglas del negocio. Es un intermediario para que los objetos no queden expuestos.

Nosotros lo utilizamos ya que tenemos controladores en nuestro modelo:

-controlador de adopción

-controlador de alarma

-controlador de animal

-controlador de cliente

-controlador de ficha medica

-controlador de seguimiento

-controlador de usuario (veterinario/ visitador)

Los mismos responden a acciones del usuario y manejamos la data que devolvemos en el controlador en un DTO.

Utilizamos este patrón ya que nos permite hacer una separación de intereses y nos da flexibilidad. La única desventaja seria que hace más difícil el testeo.

Strategy

Este patrón de comportamiento encapsula distintas formas de resolver el mismo problema en diferentes clases, permite intercambiar en momento de ejecución la forma en que un tercero resuelve un problema.

Nosotros usamos el strategy en el modelo en tres casos.

El primero fue en el recordatorio que se enviara a el cliente adoptante que podía ser enviado por WPP, SMS o EMAIL. Aquí utilizamos el patrón para que el envió se realice según la opción seleccionada ya que el algoritmo de resolución es distinto según el caso que sea.

Luego utilizamos el patrón en la parte de gestionar las alarmas, si la alarma es de un tipo tratamiento tiene una forma de manejarse, por otro lado, si es de control es diferente como se gestionará. Por lo cual acá decidimos utilizar el patrón strategy.

Por último, utilizamos el patrón strategy en el caso de la exportación de la ficha médica, la misma se puede exportar en 2 tipos de formatos distintos, por lo cual cada formato tiene su forma de gestionarse. Aquí consideramos útil el uso del patrón para poder exportar en PDF o EXCEL.

Adapter

Este patrón estructural encapsula el uso de la clase que se quiere adaptar en otra clase que concuerda con la interfaz requerida. Lo elegimos para cuando queremos seguir adelante con la implementación sin conocer exactamente como, quien y cuando resolverá una parte necesaria y solo se conoce que responsabilidad tendrá dicha parte.

Nosotros en el modelo lo usamos en varias ocasiones.

La primera en la autenticación del usuario, al ser un módulo externo que desconocemos, decidimos colocar un adapter para esa responsabilidad.

Otro caso fue el exportador de ficha médica en el cual tenemos que exportar por PDF o EXCEL, pero se pide escalabilidad por lo cual este patrón se justifica.

Luego en el envío de la notification push la utilizamos para enviarla a los veterinarios, la misma no tenemos detalles de la tercera parte por lo cual colocamos un adapter.

Por último, el recordatorio que se envía a los clientes le colocamos un adapter para el envío del mismo para SMS, EMAIL o WPP. También desconocemos detalles del tercero por lo cual se justifica el patrón.

State

Este patrón de comportamiento genera una abstracción por cada posible estado que pueda tener un objeto, define transiciones entre los posibles estados.

En el modelo nosotros lo aplicamos en dos casos.

En el primero en el estado de la adopción del cliente, ya que la adopción puede ser habilitada o no habilitada según ciertas condiciones y la acción que activa esto es cuando se realiza la adopción.

El segundo caso que lo utilizamos fue en el estado de la alarma la cual puede estar completada o estar incompleta, la misma se modifica a través del método de atender la alarma.

Singleton

Es un patron del tipo creacional , asegura que existira una unica instancia de una clase y se sugiere cuando necesitamos una unica instancia de una clase y que la misma sea accesible por terceros.

Nosotros usamos el singleton en dos casos puntuales.

El primero es la instancia del Scanner de java , la misma necesita estar instanciada una unica vez.

Segundo en los controladores , todos los queremos instanciar una unica vez por ejecucion por lo cual utilizamos este patron.