Sistema de adopción y control de animales

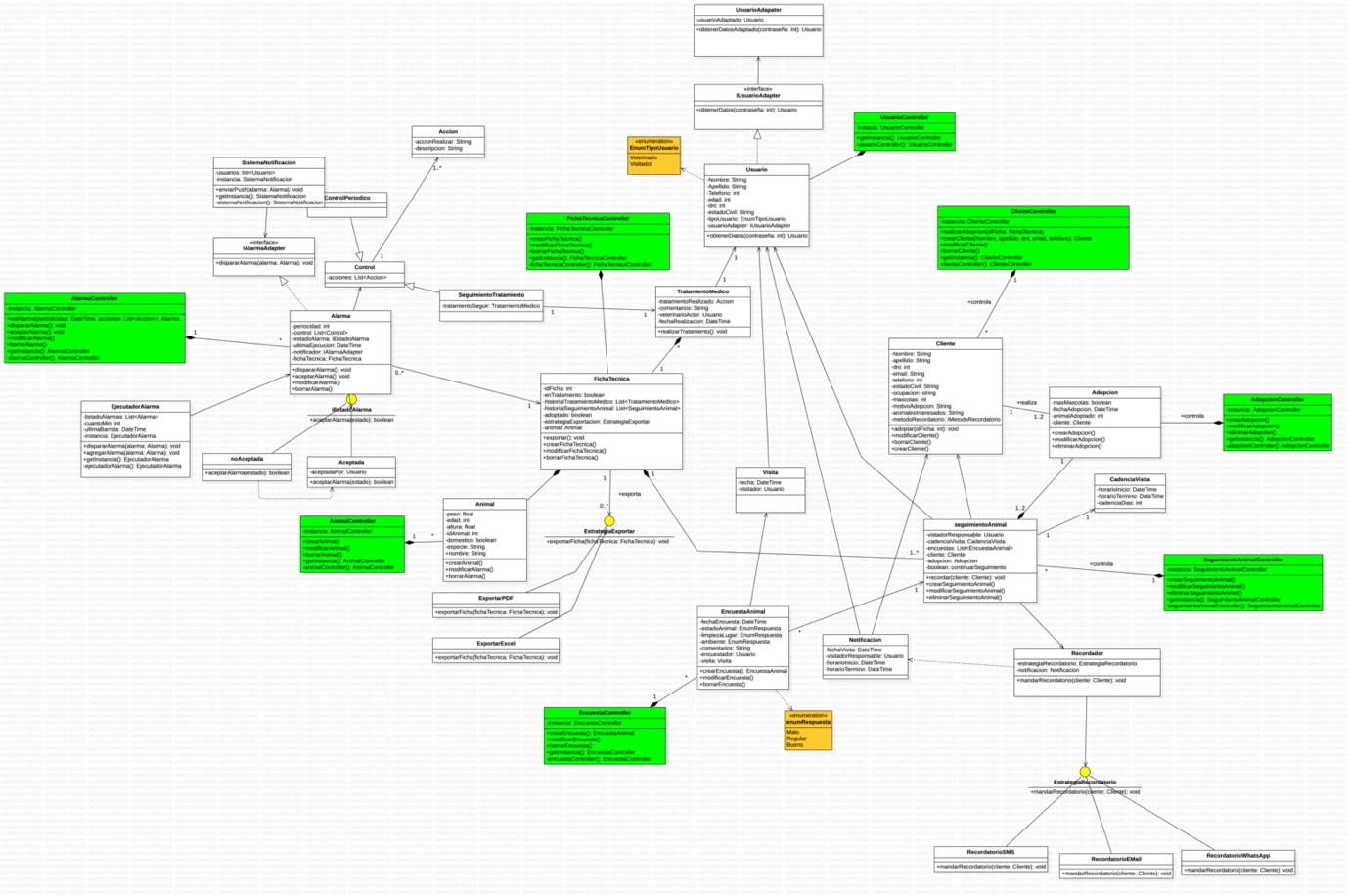
Profesor/a

* Stricagnoli, Matías Damián

Integrantes

* Mendieta, Juan Ignacio – LU: 1135071
* Barros, Matías Andrés – LU: 1058241
* Campo, Ian Brian – LU: 1124442
* Von Elm, Lucas – LU: 1133639

# Foto del diagrama de clases



# Explicación de patrones utilizados

Patrón adapter para conectar alarma a Sistema de Notificación:

Se hizo uso de un adapter que la alarma se pueda conectar a través de un método único que no cambie al sistema de notificación, y luego este reciba los parámetros necesarios a través del método y realice los necesarios por su parte, sin agregar complejidad a la clase Alarma.

Patrón Estado para el estado de la alarma:

Se usa un state pattern para cambiar el estado de la alarma de “No aceptada” a “Aceptada”, definiendo así si la alarma fue aceptada previamente y permitir también en un futuro, si más estados son necesarios, agregarlos sin complejidad alguna.

Patrón Singleton en EjecucionAlarma y SistemaNotificacion:

Cada una de estas clases requieren conocer todas las alarmas y usuarios creados respectivamente, y solo debería haber una sola instancia de cada uno de ellos al momento de ejecutarlas, por lo cual hacerlas Singleton a esas clases nos ayuda a resolver esta cuestión.

Patrón Singleton para los controllers:

Un singleton en los controllers es una práctica común, ya que nos siempre necesitamos manejar las clases con un único controlador correspondiente, por lo cual asegurarnos de que exista una única instancia es la solución que obtenemos del Singleton.

Patrón Estrategia para Exportar Ficha:

Utilizamos un patrón estrategia al momento de seleccionar el modo en que se exporta la ficha para realizar fácilmente a través de un único método el llamado a la clase correspondiente, pudiendo dejar la implementación de cada una en su clase y evitando en la clase FichaTecnica tener que agregar complejidad innecesaria para cada tipo de exportación. Además, nos permite en un futuro cambiar la forma en que funciona cada estrategia o agregar/quitar nuevas.

Patrón Estrategia para el Método de Recordatorio:

Como en el caso anterior, el patrón estrategia nos permite seleccionar fácilmente que manera desea el cliente recibir el recordatorio, encapsulando la implementación de cada método en su clase correspondiente, y haciendo que el sistema sea fácil de escalar y actualizar en un futuro.

Patrón Adapter para conectar el Usuario a la Base de Datos:

En el problema dado el Usuario se loguearía realizando un llamado a la API del módulo de base de datos (fuera del sistema), por lo cual se utiliza un adapter para poder mantener la lógica necesaria en una clase separada, permitiendo en un futuro cambiarla según sea necesario sin afectar a la clase del modelo.

# Conclusiones

El desarrollo del sistema nos sirve como un ejemplo de la importancia de los patrones como una parte fundamental de la programación. Varias partes de este resultarían imposibles de mantener o expandir sin la utilización de estos.

No solo nos ayuda a nosotros al momento de desarrollar el sistema, sino también a las futuras personas que estén encargadas de actualizar el mismo con las nuevas demandas que se tenga del mismo.

Creemos que con el estado de la informática y el desarrollo hoy en día acelerando exponencialmente, el uso de patrones y buenas prácticas resulta algo primordial al momento de pensar cada parte de un sistema. Sin ellos resulta costoso, tanto en horas-hombre como en dinero y otros recursos, el desarrollo y mantenimiento de los nuevos sistemas para abarcar las demandas del mercado.