## Single Responsabiliry

## Descripción / Explicación

El principio de responsabilidad única explica que solo debe haber una razón para tener que cambiar el código, por lo tanto, una práctica fundamental es dividir las funcionalidades agrupándolas coherentemente según lo que deban hacer. En lugar de agrupar todo en una misma clase.

## Explicación del contexto

El ejemplo es la implementación de un almacén, en un principio podemos ir pensando en las funcionalidades y las características que puede tener uno, por ejemplo, debe ser capaz de saber los productos de los que dispone, venderlos, añadir, vigilarlos con las cámaras, en un almacén trabajan personas con diferentes responsabilidades… Todo esto podemos implementarlo en la misma clase, pero hacerlo de esta forma lo único que haría sería complicarlo, solaparlo todo, hacer menos legible el código, lo convierte en un código muy sensible al cambio además de otros inconvenientes. Por eso, es necesario dividir de tal forma que recoja las funcionalidades lógicas juntas.

En nuestro caso hemos dividido lo que es el propio almacen, y ahí solo hemos incluido las funcionalidades típicas, de manejo de suministros. Por otro lado, hemos dividido los empleado en dos tipos, los vigilantes y los operadores, ambos heredan de la clase abstracta empleado. Cualquier empleado tiene unas características comunes y por lo tanto pueden ser manejados mediante una abstracción que hemos llamado EmployerManager.

## Ventajas

En el caso de que alguna de las clases deba sufrir un cambio, el programa será mucho más robusto y no estará tan solapado, permitiendo dichas modificaciones de forma fácil y eficiente.

## UML