PRINCIPIOS SOLID

HECHO CON MUCHO AMOR POR BEL REY
TWITTER: @IAMDOOMLING - IG: DOOMLINGDRAWS

SOLID es el acrónimo con el que se conoce a una serie de reglas o 'buenas prácticas' para construir código cuando trabajamos con programación orientada a objetos.

Los principios SOLID no son verdades absolutas ni leyes, simplemente son surgerencias basadas en la experiencia

Estos principios nos brindan algunos beneficios, hacen el mantenimiento del código más fácil y rápido, permiten añadir nuevas funcionalidades de forma más sencilla y favorecen la reusabilidad y calidad del código

SINGLE RESPONSIBILITY PRINCIPLE

(PRINCIPIO DE RESPONSABILIDAD ÚNICA)

Como su nombre lo indica nos recomienda que cada clase o servicio tenga una sola responsabilidad. De esta manera sí necesitamos hacer cambios en una funcionalidad es menos probable que impacte otras áreas de nuestro código.

OPEN/CLOSED PRINCIPLE

(PRINCIPIO DE ABIERTO/CERRADO)

No tiene un nombre particularmente intuitivo pero lo que este principio nos dice es que debemos escribir nuestras clases de forma que su comportamiento pueda ser extendido sin necesidad de modificar su código para manejar cada caso por separado.

Por ejemplo, si queremos tener gatos que puedan nadar, no hace falta que les demos una cola de sirena. Podemos extender su capacidad de nadar dandoles un salvavidas sin quitarles la habilidad de caminar.

Sí nos encontramos en la necesidad de manejar muchos casos lo más probable debemos volver al principio anterior y separar responsabilidades.

encontranos en teloexplicocongatitos.com

PRINCIPIOS SOLID

HECHO CON MUCHO AMOR POR BEL REY
TWITTER: @IAMDOOMLING - IG: DOOMLINGDRAWS

LISKOV SUBSTITUTION PRINCIPLE

(PRINCIPIO DE SUSTITUCIÓN DE LISKOV)

Para que nuestras clases se consideren bien diseñadas, cada instancia de una clase hija debería poder ser reemplazada por sus superclase sin que haya problemas en nuestro código. No deberíamos entonces tener hijos que no puedan cumplir las tareas de la clase base.

Si nos encontramos con este caso tenemos que replantear la herencia para que la clase base solo contenga funcionalidad útil a todas sus subclases o considerar utilizar composición en vez de herencia.

INTERFACE SEGREGATION PRINCIPLE

(PRINCIPIO DE SEGREGACIÓN DE LA INTERFAZ)

Este principio nos dice que una clase no debería recibir código que no va a utilizar. En lenguajes donde se pueden implementar interfaces esto significa que la clase debería implementar TODOS los métodos de la interface. Si esto no se cumple deberíamos considerar replantear nuestras interfaces de manera separada.

DEPENDENCY INVERSION PRINCIPLE

(PRINCIPIO DE INVERSIÓN DE DEPENDENCIAS)

Esto implica que nuestros modulos de alto nivel no deberían depender directamente de los de bajo nivel. Ambos deberían depender de abstracciones. Por ejemplo, imaginemos que nuestro gatito quiere charlar con otros animales. Deberiamos enseñarle a hablar el lenguaje de cada animal para entenderse. En cambio, bajo este principio podriamos implementar un traductor universal, de este modo ambos animales podrían conversar sin importar su especie.

Es importante recordar que estos principios son sugerencias y no necesariamente van a servir en cada proyecto, pero es bueno conocerlos para poder tomar decisiones informadas

encontranos en teloexplicocongatitos.com