

# CAPÍTULO 3

PRINCIPIOS SOLID

# SOLID

Es uno de los acrónimos más famosos en el mundo de la Programación. Introducido por Robert C. Martin a principios del 2000, se compone de 5 Principios de la programación orientada a objetos.

## PRINCIPIO DE LA RESPONSABILIDAD ÚNICA (S)

El principio nos dice que una clase debe contener una única funcionalidad, caso contrario habrá que separar las clases en múltiples clases. De esta forma conseguimos que la clase sea más entendible y fácil de mantener.

# PRINCIPIO DE SER ABIERTO Y CERRADO (O)

Este principio nos indica que nuestro código debe estar abierto a extensiones y cerrado a modificaciones, es decir, que un código ya escrito y finalizado no se toque, ya que podría afectar a su funcionamiento. Sin embargo, dado que nuestro sistema evolucionará con el tiempo, debe estar abierto a cambios, es decir, que podamos extender las clases por medio de clases abstractas, que nos permitan crear nuevos objetos que hereden de estas clases, sin alterar su funcionamiento.

Ejemplo:

```
Public abstract class figura {...}
```

↓ Herencia

```
Public class Triangulo extends figura {...}
```

# PRINCIPIO DE SUSTITUCION DE LISKOV (L)

El principio nos dice que toda clase que extienda la funcionalidad de una clase base, debe hacerlo sin alterar su funcionamiento. De esta forma, cualquier subclase que herede de la clase padre podrá ser sustituida por otra subclase, sin afectar el comportamiento de la clase padre. Si una clase hija no implementa una propiedad o método de la clase padre, la hija no puede ser subclase de la clase base, puesto que estaría violando el principio de sustitución de Liskov.

## Ejemplo

```
Public abstract class figura {...}
```

```
Public class Triangulo extend figura {...} → Subclase de  
figura
```

```
Public abstract class figura {...}
```

```
Public class circulo {...} → No puede ser subclase
```

## PRINCIPIO DE SEGREGACION DE INTERFAZ (I)

El principio nos dice que no debemos tener métodos no implementados en nuestras clases extendidas, muchas veces se da el caso que tenemos métodos que nunca implementamos. Para evitar esto, nos recomienda dividir la interfaz en interfaces más pequeñas, así todos los métodos que implementemos tendrán su propósito.

## PRINCIPIO DE INVERSION DE DEPENDENCIA (D)

Este principio establece que, en el dominio de nuestro sistema, debemos utilizar interfaces o abstracciones, elementos que cambien con poca frecuencia, de tal forma que sean las concreciones de menor las que dependan de elementos y no a la inversa.