**Universidad Autónoma de Nuevo León.**

**Facultad de Ciencias Físico Matemática.**

[**Licenciatura en Seguridad en Tecnologías de Información**](http://www.uanl.mx/oferta/licenciatura-en-seguridad-en-tecnologias-de-informacion.html) **Profesor: Lic. Miguel Angel Salazar Santillán**

**Alumna: Sofía Esmeralda Pecina Rojas.**

**Matricula: 1667093  
  
Unidad de aprendizaje: Diseño orientado a objetos  
  
tema: ¿Qué es S.O.L.I.D?**

**San Nicolás de la garza 28 de marzo del 2017**

# ¿Qué es S.O.L.I.D?

Robert C. Martín nos propuso en el año 2000 unos principios para que nuestro código orientado a objeto sea más flexible. Estos principios están recogidos sobre el acrónimo **SOLID**.

Hace mención a “aplicaciones sólidas”, donde lo de sólidas se refiere a que en el desarrollo se aplican estos principios.

A continuación que significa cada letra:

## **Single Responsability Principle o Principio de Responsabilidad Única**

Este principio indica que una clase debe tener una única responsabilidad. O como indica Rober C. Martín: “Una clase debería tener una y sólo una razón para cambiar”

## **Open-Closed Principle o Principio de abierto-cerrado**

Una entidad debe estar abierta a extensiones, pero cerrada a modificaciones, con esto, también evitamos que clases que depende de otras sean modificadas y se extienda esa modificación. Cuando se habla de extender la clase se puede usar varios métodos como la herencia o la inyección de dependencias.

## **Liskov Substitution Principle o Principio de Sustitución de Liskov**

Este principio fue creado por **Barbara Liskov** y habla de la importancia de crear todas las clases derivadas para que también puedan ser tratadas como la propia clase base. Cuando creamos clases derivadas debemos asegurarnos de no re implementar métodos que hagan que los métodos de la clase base no funcionases si se tratasen como un objeto de esa clase base.  
(Las clases derivadas deben poder ser sustituibles por sus clases base).

## **Interface Segregation Principle o Principio de Segregación de Interfaces**

Los clientes no deben ser forzosamente dependientes de las interfaces que no utilizan o de otra forma, que las interfaces no sean tan pesadas que obliguen a las clases que las implementan a implementar métodos que no necesita.

## **Dependency Inversion Principle o Principio de Inversión de Dependencia**

El objetivo de este principio es el uso de abstracciones para conseguir que una clase interactúe con otras clases sin que las conozca directamente. Es decir, las clases de nivel superior no deben conocer las clases de nivel inferior. Dicho de otro modo, no debe conocer los detalles. Existen diferentes patrones como la inyección de dependencias o service locator que nos permiten invertir el control

### Bibliografía:

<https://www.genbetadev.com/paradigmas-de-programacion/solid-cinco-principios-basicos-de-diseno-de-clases>

<http://jmperezramos.net/programacion/principios-solid-en-la-programacion-orientada-a-objetos/>