UNIVERSIDAD AUTONOMA DE NUEVO LEÓN

Facultad de Ciencias Físico Matemáticas

Lic. en Seguridad de las Tecnologías de la Información

Diseño Orientado a Objetos

*“S.O.L.I.D”*

**Nombre:** Jorge Samuel Ruan Monsivais

**Matricula:** 1736099

**Grupo:** 007

**Aula:** 409

**Maestro:** Miguel Salazar

**S.O.L.I.D**

Es un acrónimo inventado por Robert C.Martin para establecer los cinco principios básicos de la programación orientada a objetos y diseño. El objetivo de tener un buen diseño de programación es abarcar la fase de mantenimiento de una manera más legible y sencilla así como conseguir crear nuevas funcionalidades sin tener que modificar en gran medida código antiguo.

* **S - Responsabilidad simple (Single responsibility)**

Este principio indica que una clase debe tener una única responsabilidad. O como nos indica Rober C. Martín: “Una clase debería tener una y sólo una razón para cambiar”. Un ejemplo de esas clases que “hacen de todo”, es aquella que engloba una conexión a un web service, un parseador de objetos json a modelos y métodos para guardar esos objetos en base de datos, todo esto, es una misma clase. Aunque nos parezcan clases sencillas, clases pequeñas, una clase debería tener una única responsabilidad.

* **O - Abierto/Cerrado (Open / Closed)**

Una entidad debe estar abierta a extensiones pero cerrada a modificaciones. Este principio va acorde con el principio anterior ya que como se ha indicado, una clase nunca debería cambiar y si un requisito cambia, lo que debemos hacer es extender dicho comportamiento añadiendo código, no modificándolo. Con esto, también evitamos que clases que dependen de otras sean modificadas y se extienda esa modificación.

* **L - Sustitucion Liskov (Liskov substitution)**

Este principio fue creado por Barbara Liskov y dice que cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

* **I-Segregacion del interface (Interface Segregation Principle)**

Establece que los clientes de un programa dado sólo deberían conocer de éste aquellos métodos que realmente usan, y no aquellos que no necesitan usar. El ISP se aplica a una interfaz amplia y complejo para escindirla en otras más pequeñas y específicas, de tal forma que cada cliente use sólo aquella que necesite, pudiendo así ignorar al resto. A este tipo de interfaces reducidas se les llama "interfaces de rol".

* **D - Inversión de dependencias (Dependency inversion)**
* Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
* Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

La inversión de dependencia explica que una clase concreta A, no debe depender directamente de otra clase B, sino de una abstracción de B. Tal abstracción es una interfaz o una clase que sirve de base para un conjunto de clases hijas.