

**Tema:** Principios SOLID

**Curso:** Algoritmica II.

**Profesor:** Gustavo Coronel Castillo.

**Autores:**

- Huaccha Estela Jose Christian

-Reyes Calderòn Angie

-Ruiz Rafael Hernando

-Aldude Tipula Marianne

Fecha de entrega:

06/10/2017

**Indice:**

1. Resumen…………………………………………………………..….3

2. Antecedentes…………………………………………………….......4

3. Principios SOLID……………………………………………………..5

3.1. Publicaciones………………………..…………………………..6

3.2. Conceptos Importantes Elaborados….…..............................6

3.3. ¿Que es Solid?......................................................................7

3.4 Inicial Acronimo del Contexto……………..…………………....8

4. Casos…………………………………………………………………..9

4. Conclusiones………………………………………………………….9

5. Recomendaciones……………………………………………………9

6. Bibliografía………………………………………………………….....9

7. Anexos…………………………………………………………………10

**Resumen:**

Los Principios Solid son cinco principios fundamentales, uno por cada letra, principalmente la mayoria de los que inician en la programacion normalmente nunca han hablado u oido de los principios solid o la llamada programacion con codigo limpio, este principio fue creado como acronimo por Robert C. Martins. El objetivo de este principio es que una clase no conozca directamente a la otra.  
Este acronimo esta mencionado en las siguientes lineas

**S: Principio de Responsabilidad Única** (Single Responsability Principle)  
"Un objeto debe tener una única cosa"

**O: Principio de Abierto / Cerrado** (Open / Closed Principle)  
"Tenemos que ser capaces de modificar u ampliar nuestras clases sin necesidad de modificar su código"

**L: Principio de Sustitución de Liskov** (Liskov Substitution Principle)  
"Los objetos de un programa deben poder reemplazarse por instancias de sus subclases sin alterar la correctitud del programa"

**I: Principio de Segregación de Interfaces** (Interface Segregation Principle)  
"Es preferible y recomendable usar muchas interfaces específicas de cliente que una interfaz de uso general"

**D: Principio de Inversión de Dependencias** (Dependency Inversion Principle)  
"Debemos depender de las abstracciones y no de las concreciones"

**Antecedentes:**

Los antecedentes comunican al investigador los motivos que despertaron su interés en el tema. Muestra la necesidad de estudio, la situación problemática o los sucesos históricos que lo motivaron a desarrollar la investigación.

Vivimos en un mundo totalmente gobernado por la tecnología. Hoy en día, es raro encontrar a alguien que no use un ordenador en su día a día o que no tenga un Smartphone. Teniendo en cuenta el impacto que tiene la tecnología en nuestras vidas, durante estos últimos años estamos viendo varias campañas con el objetivo de promocionar la enseñanza de la programación entre todo el público, sin importar edad, sexo u ocupación.

Nosotros elegimos el  tema de investigación los principios SOLID por el simple hecho en saber en qué nos ayuda.

El objetivo de aplicar estos cinco principios es crear un código limpio y altamente mantenerlo a través de la estructura con la que este está escrito

Nosotros coincidimos que todos deberíamos de usar SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversión), ya que nos ayuda a mejorar a la hora de programar y sobre todo orientada a objetos.

**Principios Solid:**

Nació en 1952, en Palo Alto, California, él vivió cerca de Chicago durante los últimos 55 años. Fue ahí donde escribió su mi primera línea de código a la edad de 12 años, y nunca lo miró atrás. Mi primer trabajo fue de programador COBOL, pero rápidamente cambié un ensamblador para mini-computadoras a mediados de los 70.

Luego C a mediados de los 80, y C + + en los años 90. Me especialicé en sistemas de telecomunicaciones integradas. A sus 90 años se convertío en consultor, y lo llevo felizmente en su propia empresa a mí mismo desde entonces.

Fundé la empresa Object Mentor Inc. y la dirigió durante 15 años. Ofrecimos formación en ágil y orientación a objetos y consultoría para el mundo en general. Hoy en día, junto a su hija, Ángela, llevo "Tío Bob Consulting" LLC (cleancoder.com), a través de la cual continuaremos ofreciendo capacitación y consultoría en prácticas de desarrollo ágil y OO.

Ambos crearon los videos "clean code" y cleancoders.com. Y con tan solo decirles que es lo más divertido que hizo en su vida profesional. No solo se puso a escribir y actuar en sus propias películas, también ha podido comprar todos los accesorios que siempre ha querido, como modelos de naves espaciales, pistolas de rayos, "trucos de minecraft", escupideras, etc.

**Publicaciones:**

Martin, Robert Cecil (2002). Desarrollo Agile de Software: Principios, Patrones y Prácticas. Upper Saddle River, NJ: Educación de Pearson.

Martin, Robert Cecil (2009). Código limpio: un manual de artesanía ágil de software. Upper Saddle River, Nueva Jersey: Prentice Hall.

Martin, Robert Cecil (2011). The Clean Coder: Un código de conducta para programadores profesionales. Upper Saddle River, Nueva Jersey: Prentice Hall.

Martin, Robert Cecil (2017). Arquitectura Limpia: Guía de un Artesano sobre Estructura y Diseño de Software. Prentice Hall.

**Conceptos importantes elaborados:**

Principios sólidos

Prioridad de Transformación en TDD. "A medida que las pruebas se vuelven más específicas, el código de producción se vuelve más genérico".

Arquitectura limpia (una variante de "Hexagonal architecture" de Alistair Cockburn)

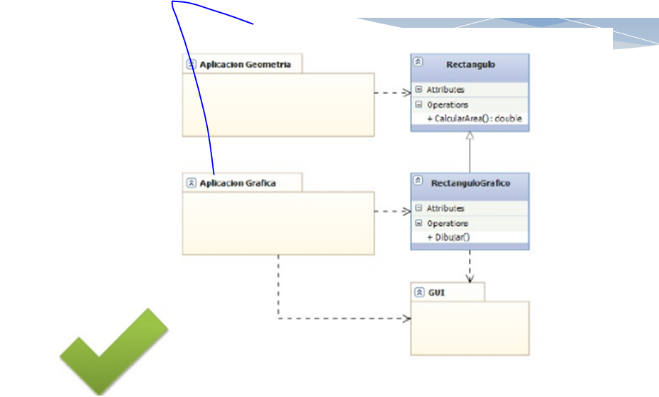
Artesanía de software

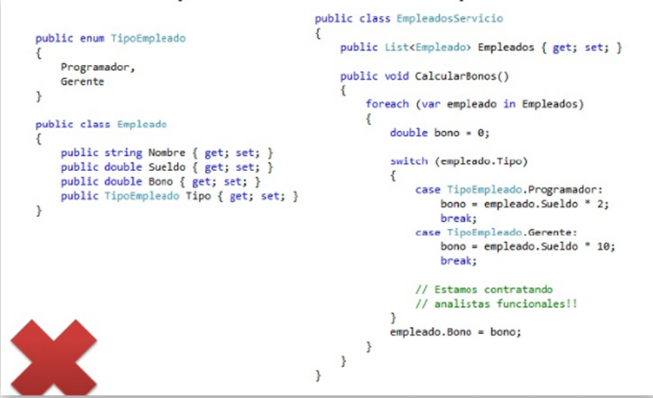
Historia de la computación El futuro de la programación.

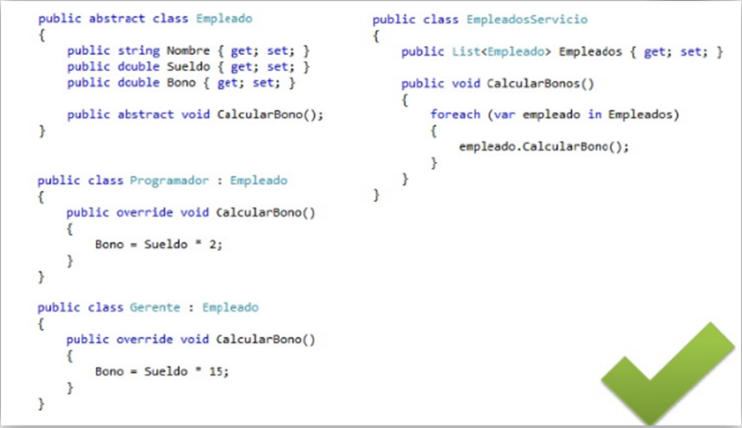
# **¿Qué es el solid?**

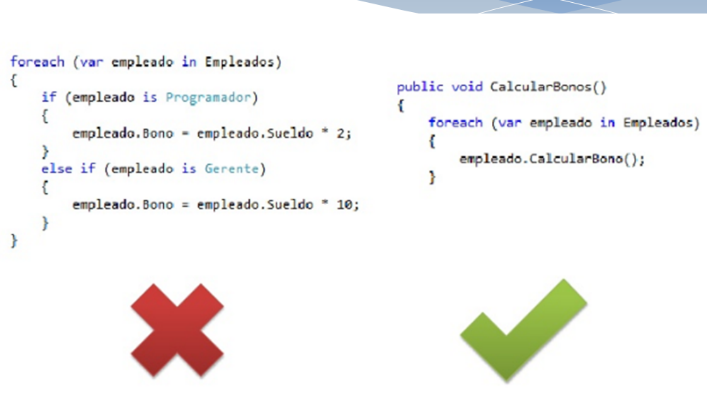
* Es acrónimo mnemónico
* Esta introducido por Robert c. Martin a comienzos de la década del 2000.
* Son cinco principios básicos de la programación orientada a objetos y el diseño.
* Ayuda a desarrollar un software de calidad, legible, entendible y fácilmente testeable.

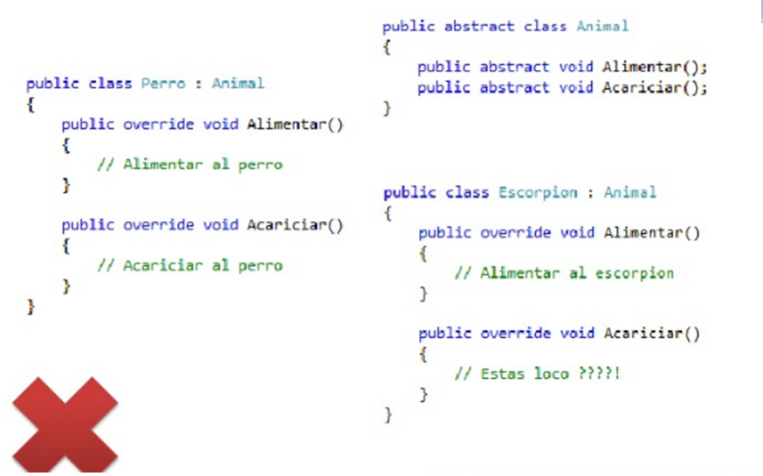
|  |  |  |
| --- | --- | --- |
| IINICIAL | ACRONIMO | CONCEPTO |
| S | **SRP** | Principio de responsabilidad única (Single responsibility principle) |
| O | **OCP** | Principio de abierto/cerrado (Open/closed principle) |
| L | **LSP** | [Principio de sustitución de Liskov](https://es.wikipedia.org/wiki/Principio_de_sustituci%C3%B3n_de_Liskov) (*Liskov substitution principle*) |
| I | **ISP** | [Principio de segregación de la interfaz](https://es.wikipedia.org/wiki/Principio_de_segregaci%C3%B3n_de_la_interfaz) (*Interface segregation principle*) |
| D | **DIP** | [Principio de inversión de la dependencia](https://es.wikipedia.org/w/index.php?title=Principio_de_inversi%C3%B3n_de_la_dependencia&action=edit&redlink=1) (*Dependency inversion principle*) |

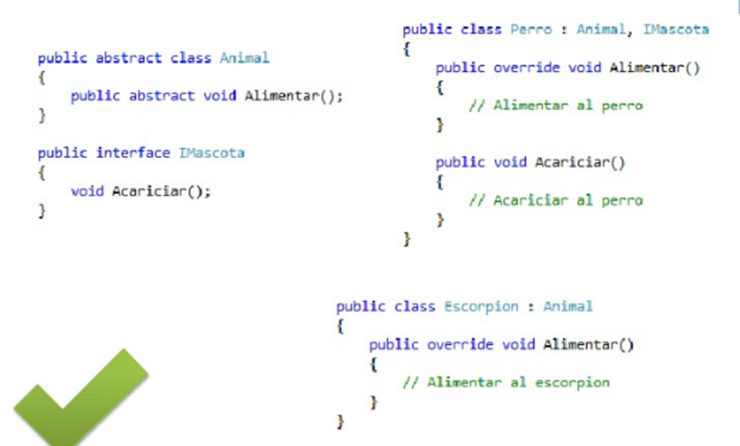
**Casos:**

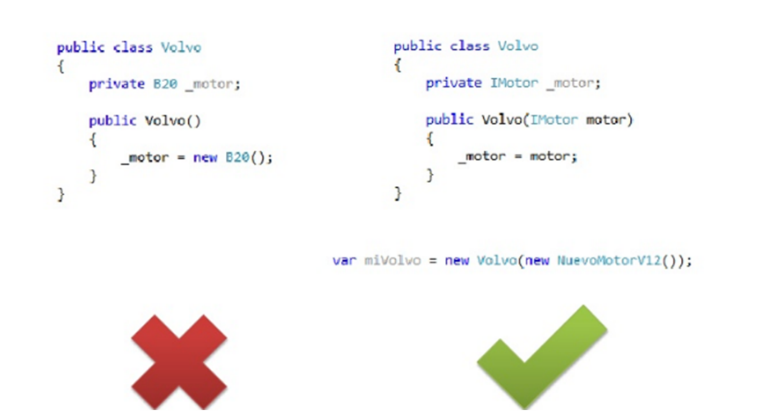












**Conclusiones:**

El principio Solid es una herramienta  que nos permite ser indispensable para proteger nuestro código frente a cambios, ya que no sólo debería haber un motivo por el cual modificar una clase.

Con la práctica, muchas veces nos encontraremos con estos límites que tendrán que ver con lo que realmente necesitamos con las complicadas técnicas de dirección. A si nuestro código nos ira dando pistas o métodos según el software evolucione con el pasar del tiempo.

**Recomendaciones:**

Se recomienda el constante uso y práctica en programación de los principios SOLID ya que son una base muy importante para obtener y generar códigos limpios, fáciles de entender y que se puedan modificar sin alterar el funcionamiento del programa.

**Bibliografía:**

De Seta, L. (2013). “Los principios SOLID para diseño de objetos”. 4 de febrero del 2013. Recuperado: https://dosideas.com/noticias/desarrollo-de-software/968-los-principios-solid-para-diseno-de-objetos

Leiva, A. (2015). “Principio de responsabilidad única (SOLID 1ª parte)”. 31 de diciembre del 2015. Recuperado: https://devexperto.com/principio-responsabilidad-unica/

Leiva, A. (2016). “Principio Open/Closed (SOLID 2ª parte)”. 7 de enero del 2016. Recuperado:   
https://devexperto.com/principio-open-closed/

Leiva, A. (2016). “Principio de sustitución de Liskov (SOLID 3ª parte)”. 14 de enero del 2016. Recuperado: https://devexperto.com/principio-de-sustitucion-de-liskov/

Leiva, A. (2016). “Principio de Segregación de Interfaces (SOLID 4ª parte)”. 21 de enero del 2016. Recuperado: https://devexperto.com/principio-de-segregacion-de-interfaces/

Leiva, A. (2016). “Principio de Inversión de Dependencias (SOLID 5ª parte)”. 28 de enero del 2016. Recuperado: https://devexperto.com/principio-de-inversion-de-dependencias/

Rubira, J. (2011). “Solid, cinco principios básicos de diseño de clases”. 14 de julio del 2011. Recuperado: <https://www.genbetadev.com/paradigmas-de-programacion/solid-cinco-principios-basicos-de-diseno-de-clases>

**Anexos:**





