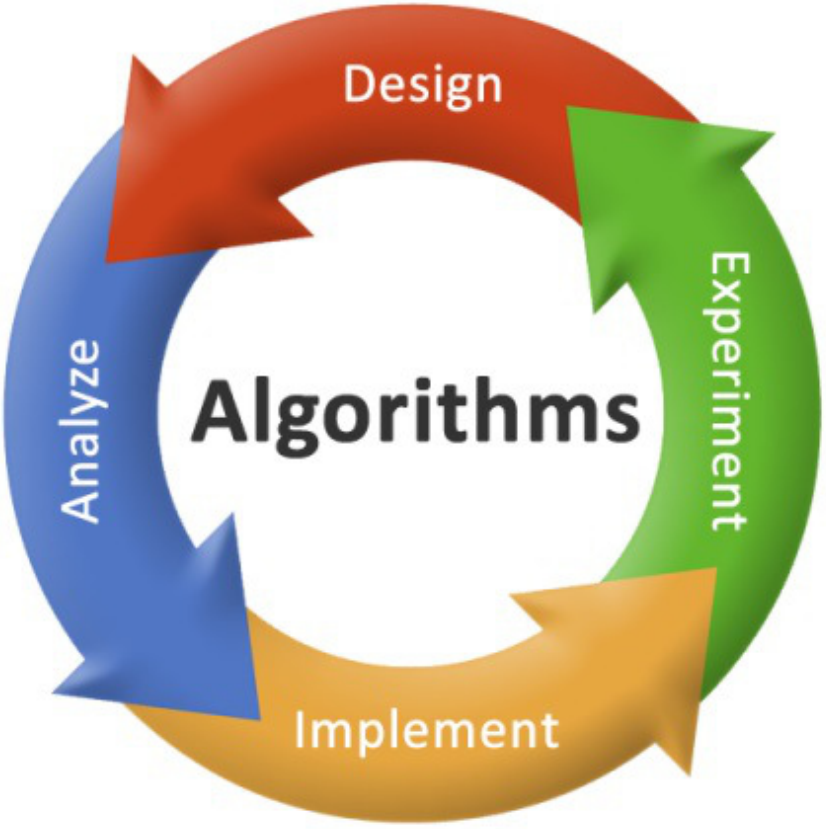
Diseño y Análisis de Algoritmos

Práctica 1 - Análisis de complejidad

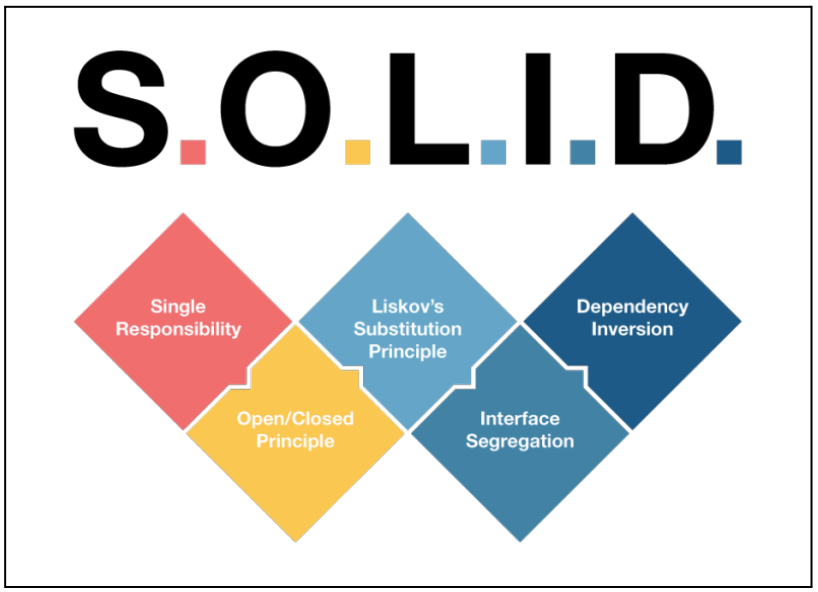


Realizado por ……. Stephan Brommer Gutiérrez
Contacto…….... alu0101493497@ull.edu.es

Universidad de La Laguna

ÍNDICE

1.- Principios SOLID



Los Principios SOLID constituyen un conjunto fundamental de directrices para el diseño de software orientado a objetos, proporcionando una base sólida para la creación de sistemas robustos, flexibles y mantenibles. A continuación, se presenta un análisis detallado de cada uno de los cinco principios:

1. Principio de Responsabilidad Única (SRP - Single Responsibility Principle):

Este principio aboga por la cohesión y la modularidad en el diseño de clases, obligando a que cada clase tenga una única responsabilidad, es decir, que cada clase tiene una única funcionalidad. Esto facilita el mantenimiento y la comprensión del código al asegurar que los cambios en una responsabilidad no afecten a otras áreas del sistema.

2. Principio de Apertura/Cierre (OCP - Open/Closed Principle):

El Principio de Apertura/Cierre promueve la extensibilidad sin modificar código existente. Al permitir que las clases se extiendan sin alterar su implementación original, se facilita la incorporación de nuevas funcionalidades sin introducir riesgos de regresión.

3. Principio de Sustitución de Liskov (LSP - Liskov Substitution Principle):

Este principio se centra en la relación entre las clases base y sus derivadas. Garantiza que las clases derivadas puedan sustituir a las clases base sin cambiar el comportamiento del programa, fomentando así la coherencia en la jerarquía de clases.

4. Principio de Segregación de Interfaces (ISP - Interface Segregation Principle):

El Principio de Segregación de Interfaces destaca la importancia de interfaces específicas para cada cliente. Evita la imposición de métodos innecesarios a las clases, mejorando la cohesión y reduciendo la complejidad. Por ejemplo, si tenemos una clase Figuras que puede representar figuras 2D y 3D, estaríamos violando este principio si creáramos un método volumen, ya que las figuras 2D no contienen un volumen.

5. Principio de Inversión de Dependencia (DIP - Dependency Inversion Principle):

Este principio aboga por la dependencia hacia abstracciones en lugar de implementaciones concretas. Al invertir las dependencias, se logra una arquitectura más flexible y resistente a cambios, facilitando la introducción de nuevas implementaciones sin afectar al código existente, es decir, que los detalles dependen de abstracciones.

2.- Patrón “Estrategia”

El patrón de diseño de estrategia es una técnica que nos permite cambiar dinámicamente el comportamiento de un objeto mediante la encapsulación de diferentes estrategias. En lugar de definir un único comportamiento estático, este patrón nos brinda la flexibilidad de elegir entre múltiples algoritmos o comportamientos en tiempo de ejecución. En el momento en el que haya más de una posible implementación de un algoritmo se debería recurrir a dicho patrón.

Siguiendo el principio de composición sobre la herencia, el patrón de estrategia se basa en la idea de separar los algoritmos del objeto principal. Esto se logra definiendo una familia de algoritmos, encapsulando cada uno y permitiendo que sean intercambiables en tiempo de ejecución. La clave aquí es permitir que el objeto principal delegue su comportamiento a una de las estrategias contenidas, proporcionando así una modularidad y flexibilidad sustanciales.