

# Diseño y Análisis de Algoritmos:

## Práctica 1

Adrián Mora Rodríguez



## ***Índice:***

<b>1. Principios SOLID</b>	<b>2</b>
<b>2. Patrón estrategia</b>	<b>3</b>
<b>3. Análisis de algoritmos</b>	<b>4</b>
<b>5. Referencias.</b>	<b>5</b>



# 1. Principios SOLID

Los principios SOLID son un conjunto de principios que, de seguirse, hacen que el código que se desarrolla sea más comprensible, legible y comprobable. Cada una de las letras de SOLID representa uno de los principios. Uno por uno son:

1. **El principio de responsabilidad única:** Una clase debe tener una única tarea y debe tener solo lo que se quiere representar, es decir, si, por ejemplo, tenemos la clase estudiante debe tener atributos propios al estudiante (DNI, nombre,...) pero no atributos relacionados (notas).
2. **El principio de apertura y cierre:** Las clases deben estar preparadas para que sea posible extenderlas pero no para que se modifiquen. Esto se refiere a que deberíamos ser capaces de agregar nuevas funciones sin modificar el código existente.
3. **Principio de sustitución de Liskov:** Una subclase no debe reducir el comportamiento sino solo debe ampliarlo o, si acaso, mantenerlo. Es por esto que este principio dice que una subclase debe poder ser sustituida por una clase base.
4. **Principio de segregación de interfaces:** Es mejor tener muchas interfaces específicas que una interfaz que realice varias funciones ya que después al realizar más interfaces a partir de ella puede darse el caso de tener que desarrollar funciones que son irrelevantes.
5. **Principio de inversión de dependencia:** El principio de apertura y cierre nos dice que las clases deben estar abiertas a extenderlas pero no a la modificación. Este principio lo complementa ya que dice que las clases deben depender de clases abstractas en lugar de clases concretas ya que ayudan a la fácil expansión de las mismas.

El principio de responsabilidad única es uno de los más importantes ya que gracias a él la claridad, mantenibilidad y flexibilidad aumenta mucho en los proyectos software. Otra característica que mejora este principio es la reutilización de código para otros proyectos ya que, como cada clase realiza una función concreta, no se arrastran funcionalidades no deseadas o inútiles en la implementación que vamos a realizar.



## 2. Patrón estrategia

El patrón estrategia es una manera de modelar y diseñar software, que proporciona una estructura flexible para manejar las distintas formas de hacer una tarea de un sistema. Este patrón implica la creación de una interfaz básica y genérica que encapsula métodos comunes a una familia de algoritmos relacionados. Cada algoritmo específico se implementa en una clase concreta que hereda de la interfaz genérica, lo que facilita la extensión y modificación del sistema.

Una de las ventajas clave de este patrón de diseño es su capacidad para permitir, de manera sencilla, la introducción de nuevos algoritmos. Al separar la implementación de los algoritmos de la lógica principal del sistema, el patrón estrategia ayuda a tener una mayor modularidad. Además, al interactuar con la interfaz genérica, los clientes pueden utilizar los diferentes algoritmos de manera transparente, sin necesidad de conocer los detalles internos de cada uno.



### 3. Análisis de algoritmos

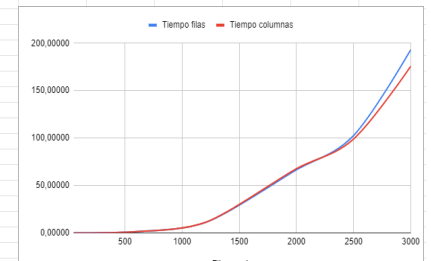
Los distintos algoritmos se han ejecutado en un ordenador con un chip M2 de Apple (CPU de 8 núcleos (4 de rendimiento y 4 de eficiencia), GPU de 8 núcleos, Neural Engine de 16 núcleos y 100 GB/s de ancho de banda de memoria), una memoria RAM de 8 GB y una memoria SSD de 256GB. Tras realizar la ejecución de los algoritmo he obtenido los siguientes resultados:

Se puede observar que el algoritmo por columnas, en la mayoría de los casos, es mejor en cuanto a tiempo de ejecución que el algoritmo por filas.

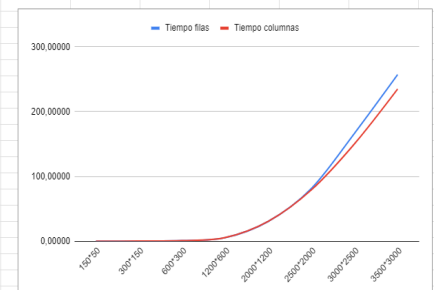
Aunque al principio estas diferencias son prácticamente indistinguibles, según se aumenten el número de filas y columnas el algoritmo por filas aumenta su tiempo de ejecución con respecto al de columnas. Este fenómeno se debe a la manera en que se gestiona la memoria y cómo se realizan los accesos a ella. Acceder a los elementos de una columna en una matriz, implica un mayor costo que acceder a los elementos de una fila. Esto se debe a que, normalmente, una fila puede ser almacenada en memoria, como un array, lo que facilita el acceso secuencial a sus elementos. Por otra parte, para acceder a los elementos de una columna, el sistema necesita realizar cálculos adicionales para determinar la ubicación de cada elemento en la memoria, ya que no están contiguos.

Este efecto se puede observar en los algoritmos de multiplicación de matrices. En el algoritmo por columnas, se modifica la fila en cada iteración en lugar de la columna, lo que aprovecha la localidad de los datos y facilita mantenerla en memoria. Debido a que traer una columna de memoria es más costoso que traer una fila, este algoritmo optimiza los accesos a memoria y, por lo tanto, el rendimiento. Para mejor visualización de las tablas usar la referencia 7.

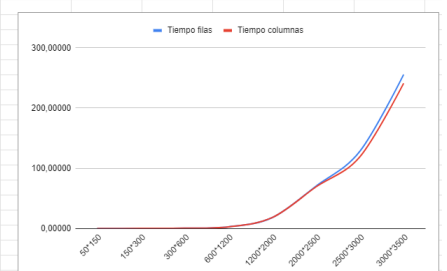
Filas y columnas	Tiempo filas	Tiempo columnas
50	0.00176	0.00148
150	0.02319	0.02006
300	0.17272	0.17029
600	1.35059	1.35156
1200	10.78820	10.87970
2000	66.52960	67.72900
2500	102.72200	99.04530
3000	193.22200	175.84100



Filas y columnas	Tiempo filas	Tiempo columnas
150*50	0.01102	0.00699
300*150	0.08478	0.08128
600*300	0.68136	0.68320
1200*600	5.38663	5.42090
2000*1200	30.50730	30.62170
2500*2000	81.41000	79.24880
3000*2500	166.62600	150.41600
3500*3000	257.18800	234.95900




Filas y columnas	Tiempo filas	Tiempo columnas
50*150	0.00493	0.00327
150*300	0.04381	0.03993
300*600	0.34275	0.33916
600*1200	2.69630	2.68821
1200*2000	18.02360	18.12770
2000*2500	70.43500	69.11030
2500*3000	128.13500	119.02000
3000*3500	255.79200	241.20500





## 5. Referencias.

- [1] <https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/>
- [2] <https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>
- [3] <https://www.freecodecamp.org/news/a-beginners-guide-to-the-strategy-design-pattern/>
- [4] <https://refactoring.guru/es/design-patterns/strategy>
- [5] [https://en.wikipedia.org/wiki/Row-\\_and\\_column-major\\_order](https://en.wikipedia.org/wiki/Row-_and_column-major_order)
- [6] <https://www.atc.uniovi.es/telematica/2ac/Transparencias/T04-Principio-de-Localidad.pdf>
- [7]  Análisis de rendimiento de algoritmos de multiplicación de matrices