

Patrones de diseño de Software

Singleton Pattern



Universidad
Autónoma
de **Coahuila**



José Eduardo Onofre Suárez

El patrón Singleton

Nombre del patrón

Singleton o patrón de *Instancia única*.

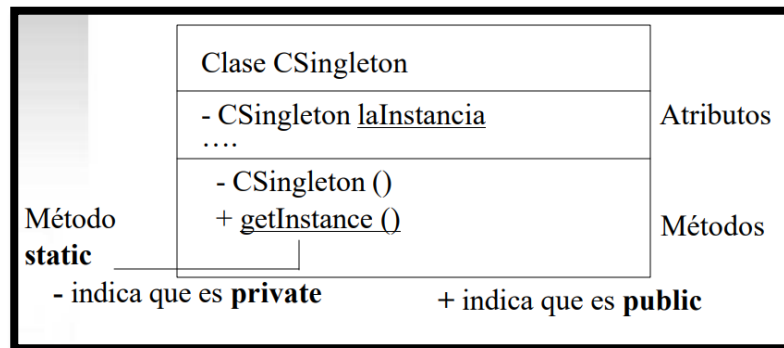
Problema

Surge a partir de la necesidad de tener solamente una y solo una instancia de un objeto. Además de requerir de un acceso global y el uso del “lazy initialization”.

Solución

El constructor de la clase debe ser **PRIVADO**.

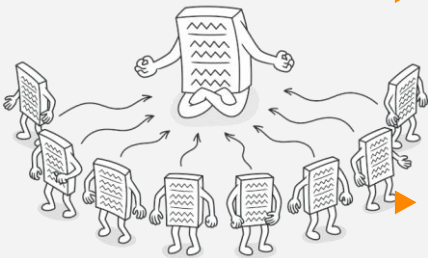
Se proporciona un método **Estático** en la clase que devuelve la única instancia de la clase.



Lazy Initialization: Es una optimización de rendimiento que consiste en posponer o retrasar la creación de un objeto hasta que este sea necesitado.

Ventajas del patrón

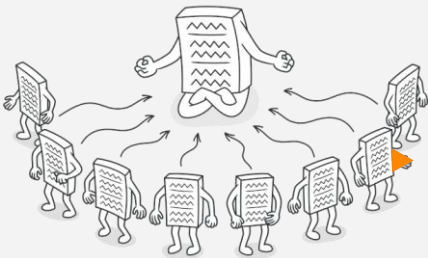
- ▶ **Acceso controlado a la única instancia:** Encapsula su única instancia, puede tener un control estricto sobre como y cuando acceden a ella los clientes.
- ▶ **Espacio de nombres reducido:** Es una mejora sobre las variables globales. Evita contaminar el espacio de nombres con variables globales que almacenan las instancias.
- ▶ **Permite el refinamiento de operaciones y la representación:** Se puede crear una subclase de la clase Singleton, y es fácil configurar una aplicación con una instancia de esta clase extendida, incluso en tiempo de ejecución.
- ▶ **Permite un número variable de instancias:** Hace que sea fácil permitir mas de una instancia de la clase. Solo se necesitaría cambiar la operación que otorga acceso a la instancia del Singleton.



Desventajas del patrón

- ▶ **Dificultad para realizar testing:** Un Singleton es como una valor de variable global. Lo global y la orientación a objeto no son buenos amigos ya que introduce un “estado persistente”, es decir valores que se mantienen siempre dificultando el uso de objeto de reemplazo(mock) en test.

- ▶ **Promociona el alto acoplamiento:** Si hay algo que debe ser alto en la orientación a objetos es la cohesión y no el acoplamiento. El Singleton es instanciado directamente desde su propia clase promocionando el uso de métodos privados y estáticos. Esto acopla la clase que los use además de impedir el uso adecuado de inyección de dependencias.



- ▶ **Restricción de ejecuciones paralelas:** Aunque un objetivo del Singleton sea la gestión de un recurso compartido esto restringe operar de forma paralela a la aplicación y lo transforma en un cuello de botella de operaciones seriales que no es recomendable cuando la demanda es alta.



UsoS

Singleton Pattern

Algunos ejemplos de uso y relación con otros patrones



The diagram consists of three overlapping circles arranged horizontally. The leftmost circle is orange and contains the text 'Logger'. The middle circle is dark gray/black and contains the text 'Configuración'. The rightmost circle is orange and contains the text 'Pool y Factory'. The circles overlap such that the middle circle is partially covered by the other two, suggesting a central role or relationship between all three.

Logger

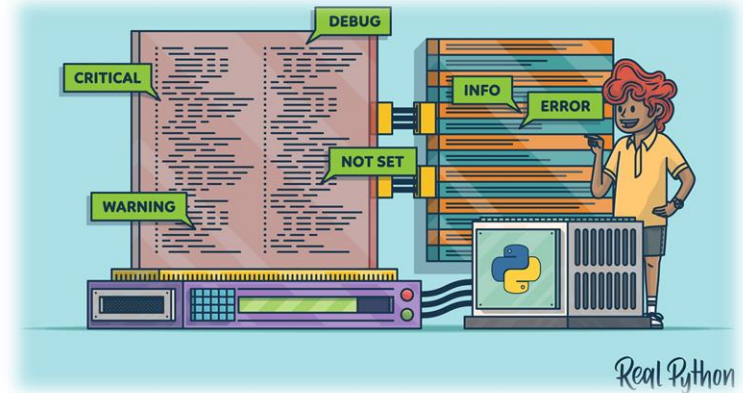
Configuración

Pool y Factory

Logger

Este es el ejemplo más adecuado de como usar un Singleton.

Tienes un archivo de log(recurso compartido) que solo debe ser accesado por un proceso. Además de no permitir la creación de nuevos objetos cada vez que realiza una operación de logging.



Configuración

Permite tener un punto de acceso único a los parámetros de configuración de una aplicación manteniendo el estado de estos durante la ejecución, independientemente de donde se cargaron los valores al iniciar el sistema, si de una base de datos o un archivo.



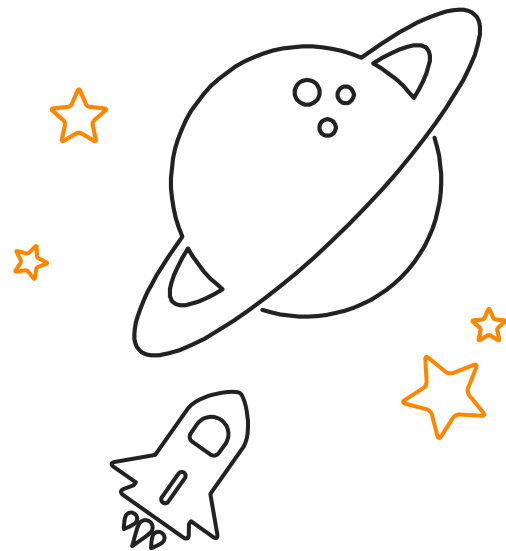
Pool & Factory

Aunque estos sean igualmente patrones de diseño, usan el Singleton como base ya que un pool o una factoría es un objeto único que tiene un estado que ofrece como servicio, permitiendo que la aplicación acceda a ese estado siempre que lo necesite.



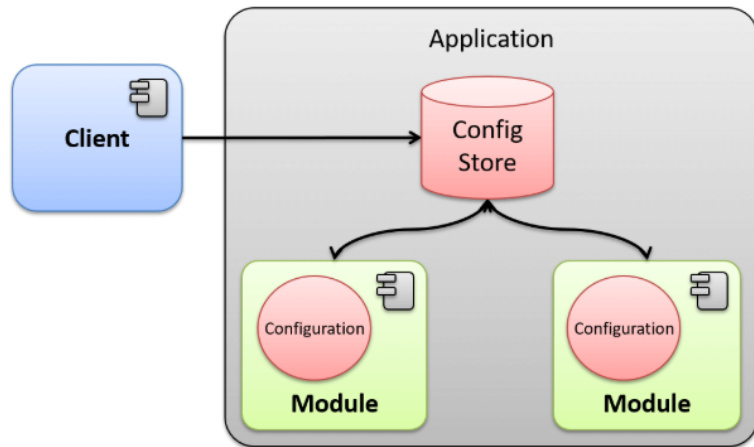
Aplicaciones

Singleton Pattern

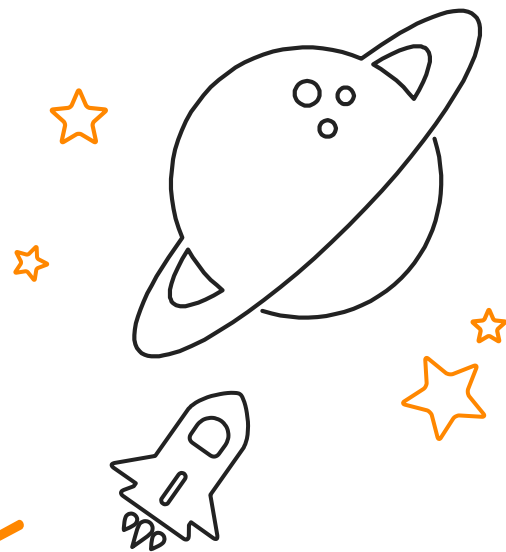


EJEMPLO DEL MUNDO REAL

Mediante la implementación del patrón de diseño *Singleton* crearemos una aplicación que permite gestionar la configuración del sistema desde un único punto centralizado. Así, cuando la aplicación inicie, cargara la configuración inicial y está disponible para toda la aplicación.



Descubre como el patrón Singleton nos ayuda a resolver este problema.



Implementación

Singleton Pattern

Diagrama UML

Los componentes que conforman el patrón son los siguientes:

- Client:** Componente que desea obtener una instancia de la clase Singleton.
- Singleton:** Clase que implementa el patrón Singleton, de la cual únicamente se podrá tener una instancia durante toda la vida de la aplicación.

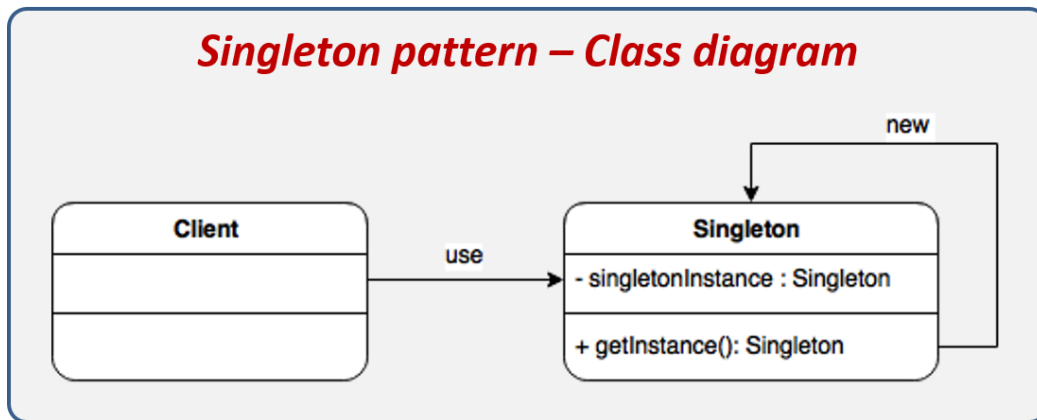
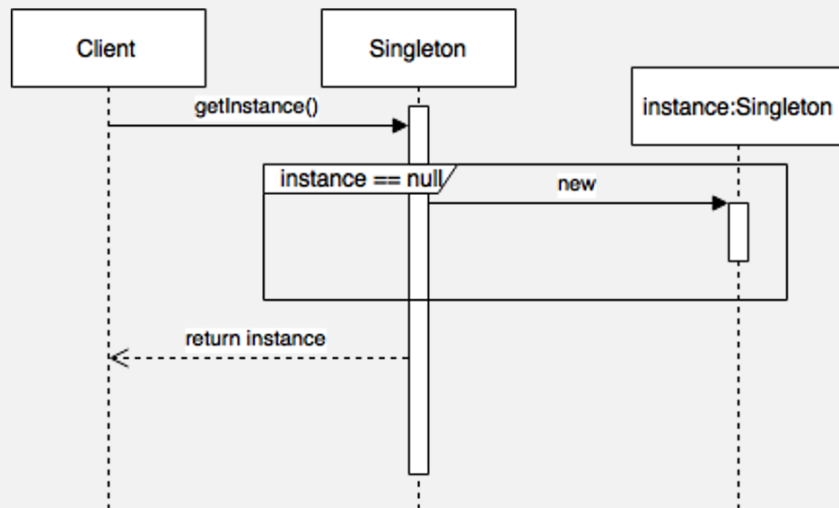


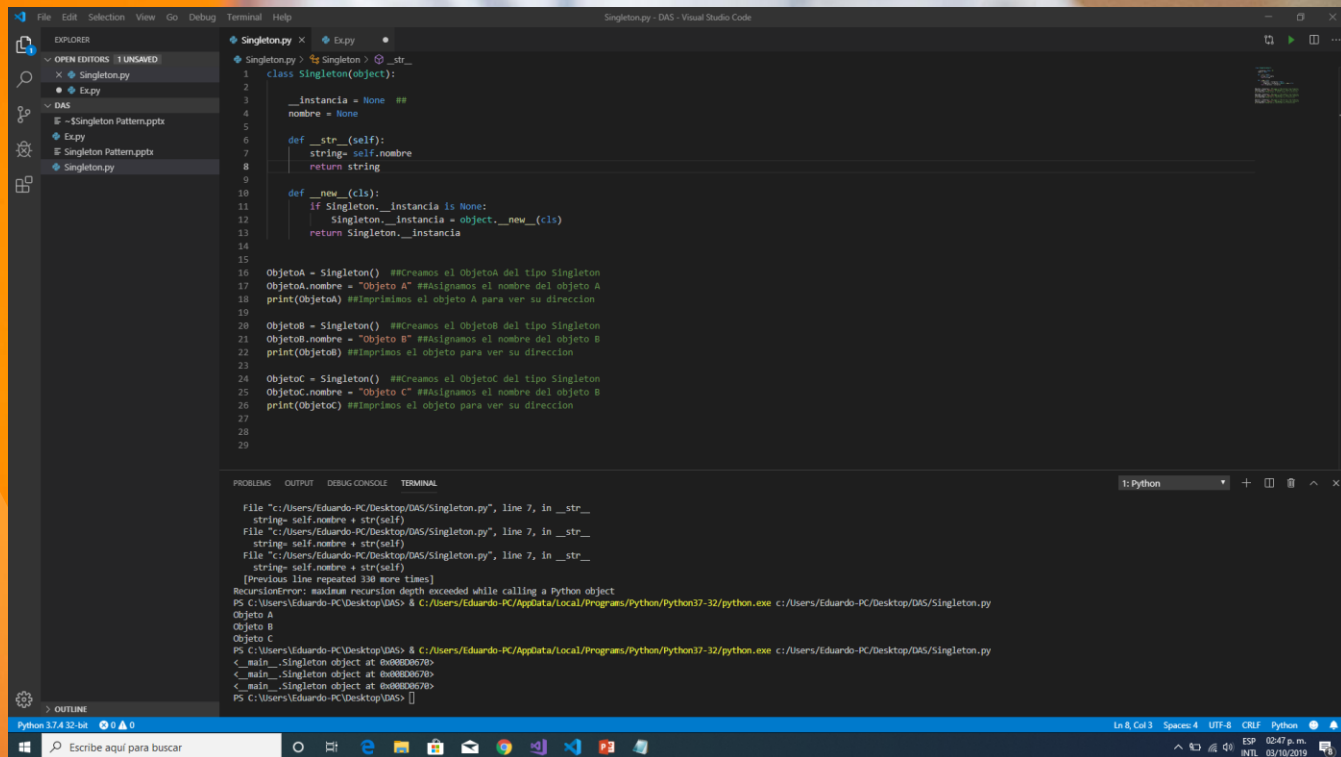
Diagrama de Secuencia

- El cliente solicita la instancia al **Singleton** mediante el método estático **getInstance**.
- El **Singleton** validará si la instancia ya fue creada anteriormente, de no haber sido creada entonces se crea una nueva.
- Se regresa la instancia creada en el paso anterior o se regresa la instancia existente en otro caso.

Singleton pattern – Diagram of sequence



Código en Python



```
File Explorer
OPEN EDITORS (1 UNSAVED)
Singleton.py
Ex.py
DAS
Singleton Pattern.pptx
Ex.py
Singleton Pattern.pptx
Singleton.py

Singleton.py
class Singleton(object):
    __instancia = None
    nombre = None

    def __str__(self):
        string = self.nombre
        return string

    def __new__(cls):
        if Singleton.__instancia is None:
            Singleton.__instancia = object.__new__(cls)
            return Singleton.__instancia

ObjetoA = Singleton()
ObjetoA.nombre = "Objeto A"
print(ObjetoA)

ObjetoB = Singleton()
ObjetoB.nombre = "Objeto B"
print(ObjetoB)

ObjetoC = Singleton()
ObjetoC.nombre = "Objeto C"
print(ObjetoC)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

File "c:/Users/Eduardo-PC/Desktop/DAS/Singleton.py", line 7, in __str__
string = self.nombre + str(self)
File "c:/Users/Eduardo-PC/Desktop/DAS/Singleton.py", line 7, in __str__
string = self.nombre + str(self)
File "c:/Users/Eduardo-PC/Desktop/DAS/Singleton.py", line 7, in __str__
string = self.nombre + str(self)
[Previous line repeated 300 more times]
RecursionError: maximum recursion depth exceeded while calling a Python object
PS C:\Users\Eduardo-PC\Desktop> & C:\Users\Eduardo-PC\AppData\Local\Programs\Python\Python37-32\python.exe c:/Users/Eduardo-PC/Desktop/DAS/Singleton.py
Objeto A
Objeto B
Objeto C
PS C:\Users\Eduardo-PC\Desktop> & C:\Users\Eduardo-PC\AppData\Local\Programs\Python\Python37-32\python.exe c:/Users/Eduardo-PC/Desktop/DAS/Singleton.py
<_main__._Singleton object at 0x00000670>
<_main__._Singleton object at 0x00000670>
<_main__._Singleton object at 0x00000670>
PS C:\Users\Eduardo-PC\Desktop> []

Python 3.7.4 32-bit

Write here to search

Ln 8, Col 3 Spaces: 4 UTF-8 CRLF Python

ESP 02:47 p. m.
NTL 03/10/2019

Gracias!

