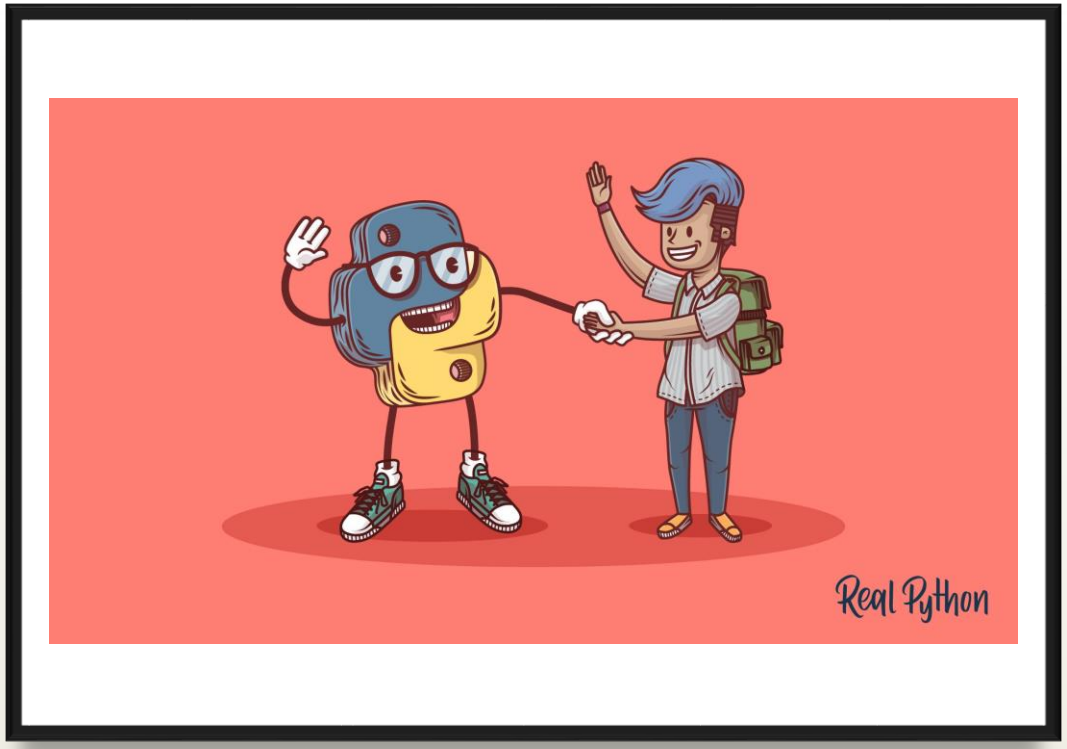
PATRÓN DE DISEÑO: SINGLETON

DISEÑO Y ARQUITECTURA DE SOFTWARE

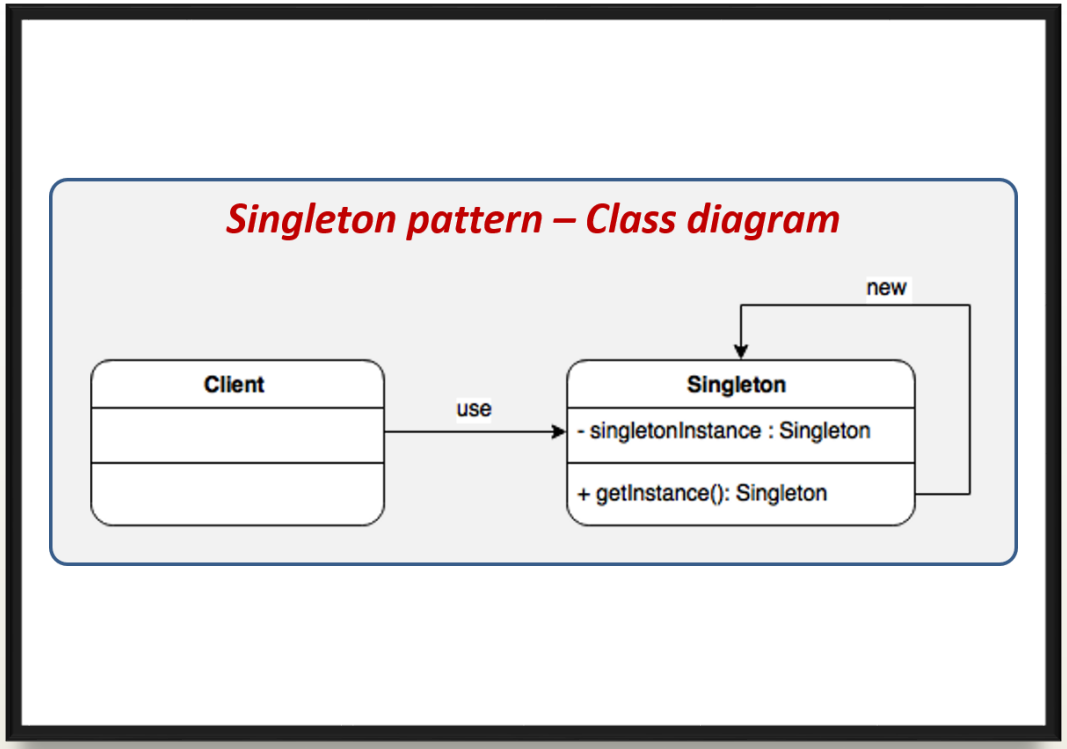
NOMBRE Y CLASIFICACIÓN THE SINGLETON PATTERN

- También es conocido como Singular o Único.
- Se clasifica como Patrón Creacional.
- Es el más simple en términos de su diagrama de clases, ya que su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.



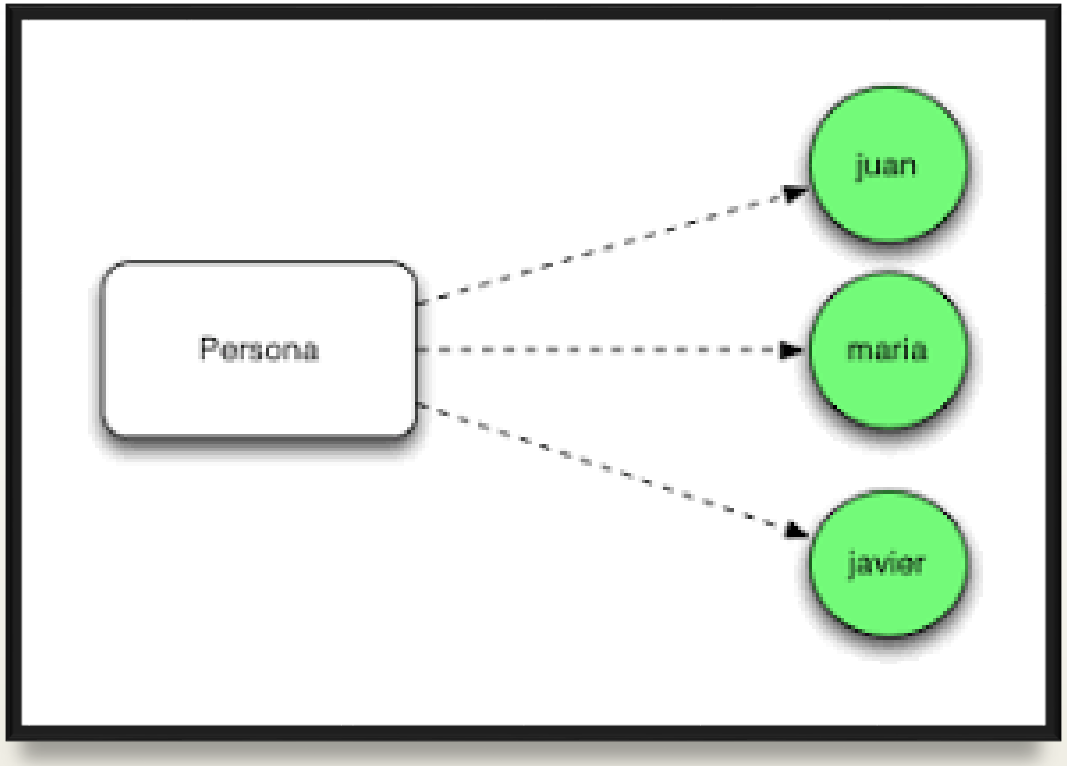
INTENCIÓN

- La intención de este patrón es garantizar que solamente pueda existir una única instancia de una determinada clase y que exista una referencia global en toda la aplicación.



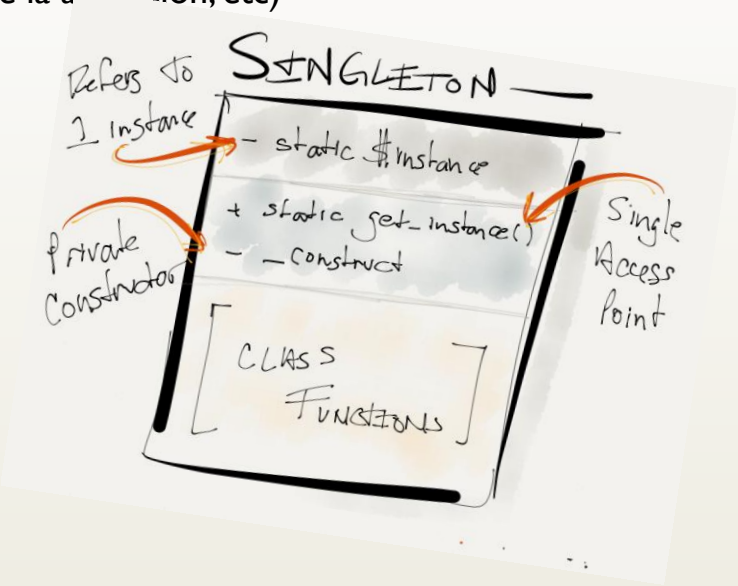
MOTIVACIÓN

- Es importante que algunas clases tengan exactamente una instancia.
- ¿Cómo lo podemos asegurar y que ésta sea fácilmente accesible?.
- Una variable global no previene de crear múltiples instancias de objetos.



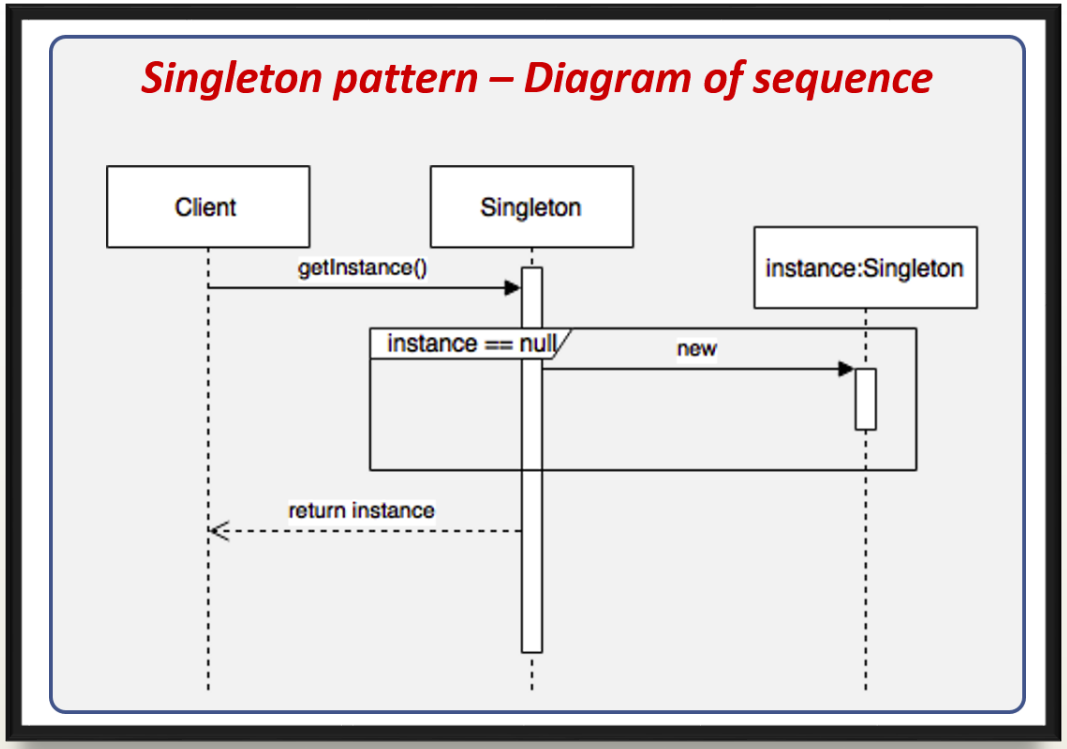
APLICACIÓN

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Administrar una conexión a una base de datos Administrador de archivos
- Recuperación y almacenamiento de información en archivos de configuración externos
- Almacenamiento de algunos estados globales (idioma del usuario, hora, zona horaria, ruta de la aplicación, etc)



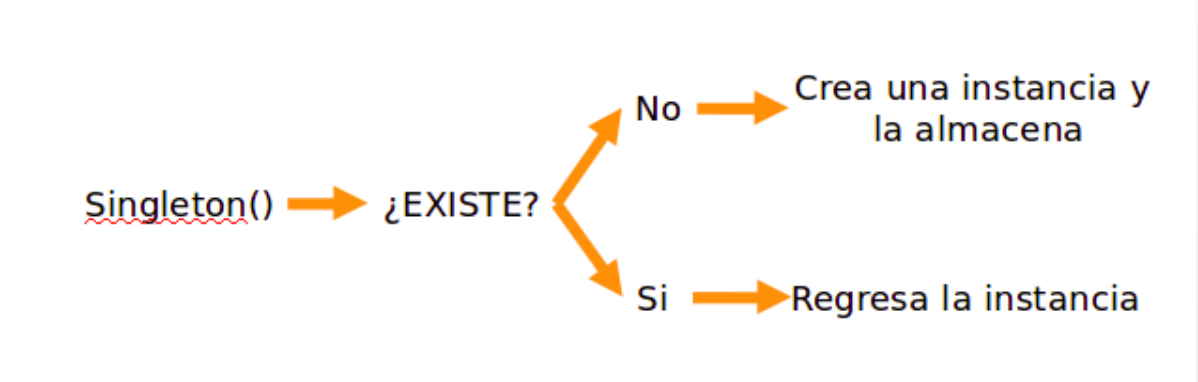
ESTRUCTURA

- 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.



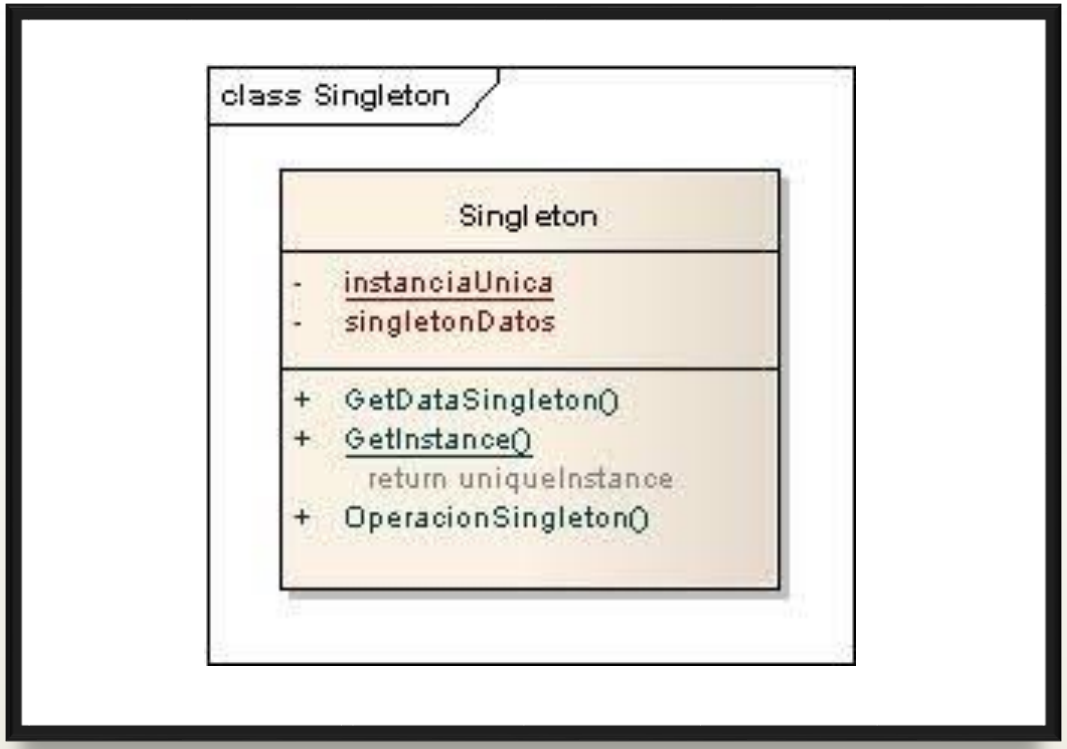
PARTICIPANTES

- **Singleton**: Define una operación Instancia que permite que los clientes accedan a su única instancia.
- **Instancia**: Es una operación de clase. Puede ser responsable de crear su única instancia.



COLABORACIONES

- Los clientes acceden a la instancia única solamente a traves de la operación **getInstance**.



CONSECUENCIAS

- **Acceso controlado a la instancia**: Debido a que la clase Singleton encapsula su instancia, puede tener un control de cuándo y cómo los clientes acceden a ella.
- **Reduce el número de variables**: El patrón Singleton evita tener variables globales que sólo guardan instancias de clases.
- **Permite cambiar el número de instancias**: Este patrón facilita que se añadan más instancias de la clase Singleton en caso de que ya no queramos solo una. Además, se puede controlar el número de instancias que se usan. Lo único que hay que cambiar en la operación instance().
- **Permite el refinamiento de operaciones y la representación**. Se puede crear una subclase de Singleton.
- **Más flexible** que las operaciones de clase (static en C#, Shared en VB .NET).

IMPLEMENTACIÓN

- De la implementación podemos destacar los siguientes aspectos que hacen referencia al patrón:
- **Instancia de la clase como constante**: Es la única forma de asegurar que solo sea una instancia para toda la aplicación.
- **Constructor de la clase privado**: Para impedir la creación de instancias de la clase fuera de ella.
- **Método getInstance**: Es la forma de asegurar la entrega de la única instancia existente de la clase.

EJEMPLOS DE CÓDIGO

USOS CONOCIDOS

- Un ejemplo del patrón Singleton es la relación entre las clases y sus metaclases. Las metaclases no tienen nombre, pero estos no pierden de vista su única instancia y no crean otra.
- Este patrón es implementado por otros patrones como **Abstract Factory, Builder** y **Prototype.**

PATRONES RELACIONADOS

- **Abstract Factory**: muchas veces son implementados mediante singleton, ya que normalmente deben ser accesibles públicamente y debe haber una única instancia que controle la creación de objetos.
- **Monostate**: es similar al singleton, pero en lugar de controlar el instanciado de una clase, asegura que todas las instancias tengan un estado común, haciendo que todos sus miembros sean de clase.

```python
class Singleton(object):
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = object.__new__(cls, *args, **kwargs)

        return cls._instance


class Ave(Singleton):
    nombre = u""

a = Ave()
b = Ave()

a.nombre = u"Cuervo"
b.nombre = u"Gorrión"
print a.nombre, b.nombre # Gorrion Gorrion
```

```
class Singleton(object):
        _instance = None

        def __new__(cls, *args, **kwargs):
                if not cls._instance:
                        cls._instance = object.__new__(cls, *args, **kwargs)

                return cls._instance

class Presidente(Singleton):
        nombre= ""

a = Presidente()
b = Presidente()

a.nombre = "Felipe Calderon"
b.nombre = "EPN"
print (a.nombre,b.nombre) #EPN EPN
```