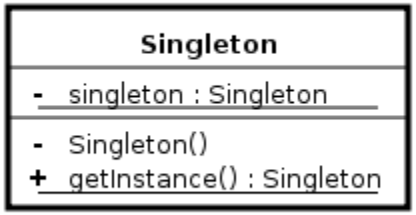
Patron: Singleton

Definición

En ingeniería de software, el patrón singleton (instancia única en inglés) es un patrón de diseño diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.



JAVA!!!

```java
public class Singleton {
    private static final Singleton INSTANCE = new Singleton();

    // El constructor privado no permite que se genere un constructor por defecto.
    // (con mismo modificador de acceso que la definición de la clase)
    private Singleton() {}

    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

JAVA!!!

```java
public class Singleton {
    private static final Singleton INSTANCE = new Singleton();

    // El constructor privado no permite que se genere un constructor por defecto.
    // (con mismo modificador de acceso que la definición de la clase)
    private Singleton() {}

    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

Pero Python no es Java

Python - Opción 1 - Delegar Llamadas

```
class OnlyOne:
    class __OnlyOne:
        def __init__(self, arg):
            self.val = arg
        def __str__(self):
            return repr(self) + self.val
    instance = None
    def __init__(self, arg):
        if not OnlyOne.instance:
            OnlyOne.instance = OnlyOne.__OnlyOne(arg)
        else:
            OnlyOne.instance.val = arg
    def __getattr__(self, name):
        return getattr(self.instance, name)
```

Python - Opción 2 - Redefinir Constructor

```
class Singleton (object):
    instance = None
    def __new__(cls, *args, **kargs):
        if cls.instance is None:
            cls.instance = object.__new__(cls, *args, **kargs)
        return cls.instance

#Usage
mySingleton1 = Singleton()
mySingleton2 = Singleton()
```

Python - Opción 3 - Metaclase

```
class Singleton(type):

    def __init__(cls, name, bases, dct):
        cls.__instance = None
        type.__init__(cls, name, bases, dct)

    def __call__(cls, *args, **kw):
        if cls.__instance is None:
            cls.__instance = type.__call__(cls, *args,**kw)
        return cls.__instance

class A:
    __metaclass__ = Singleton
    # Definir aquí el resto de la interfaz
```

Python - Opción 4 - Usar Monostate

```
class MonoState(object):
    _shared_state = {}

    def __init__(self, arg):
        self.__dict__ = MonoState._shared_state
        self.val = arg

    def __str__(self):
        return self.val
```

Java - Bonus

```java
public class MonoState{
    private static int state;

    public int getState(){
       return Monostate.state;
    }

    public void setState(int state){
       Monostate.state =  state;
    }
}
```

```java
Monostate i1 = new MonoState();
Monostate i2 = new MonoState();

System.out.println(i1.getState); # print 0
System.out.println(i2.getState); # print 0

i1.setState(2);

System.out.println(i1.getState); # print 2
System.out.println(i2.getState); # print 2
```

Python - Hay Más Formas!

- http://python-3-patterns-idioms-test.readthedocs.io/en/latest/Singleton.html
- https://en.wikipedia.org/wiki/Singleton_pattern
- https://es.wikipedia.org/wiki/Singleton_pattern (Tiene muchos ejemplos)