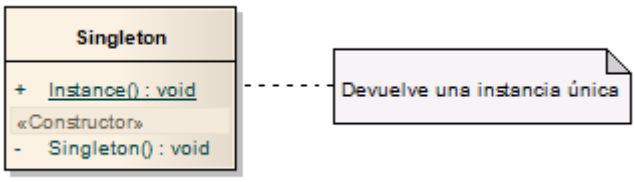
PATRÓN SINGLETON

Objetivo:

«Asegurarse de que una clase tiene una única instancia, proporcionando acceso global a ella.»



Hay clases que deben instanciarse una única vez. El acceso a un sistema de archivos, a la cola de impresión o al gestor de ventanas del sistema operativo debería realizarse por un único objeto, siendo labor de la propia clase el controlar que la instancia sea única. Por norma general, esta clase será accesible de forma global, y el proceso de instanciado no suele requerir parámetros.

Como podemos observar en el diagrama, nuestra clase *Singleton* constará al menos con dos métodos:

- Un método *Instance()* de carácter **estático** (método de clase) que se encargará de instanciar la clase.
- Un constructor **privado** que evitará que se creen nuevos objetos mediante *new()*, haciendo que el método *Instance()* sea el único que puede generar la instancia.

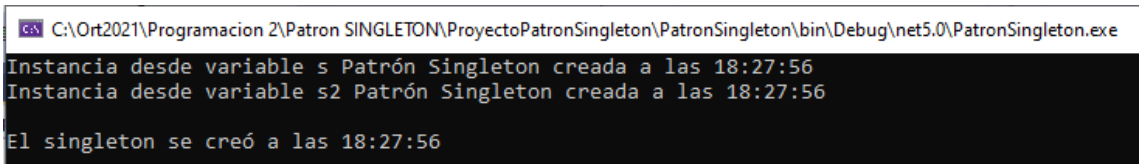
Implementación del patrón

Implementar este patrón es simple: **dado que el constructor es privado, tan sólo podrá ser invocado desde el interior de la propia clase**. Por lo tanto, el esquema del patrón se reduce a la siguiente:

- La clase *Singleton* contará con un atributo de la propia clase *Singleton*, de carácter privado.
- El constructor (privado) se encargará de construir el objeto.
- El método estático *Instance()* realizará dos operaciones:
  - Comprobar si el atributo es *null*. En ese caso, se invocará al constructor. En caso contrario, se devolverá el objeto existente.

1. En código será simple: **Ver clase Singleton.cs del proyecto PatronSingleton**

2. Usamos programa para comprobar su funcionamiento, en el que se instanciará el *Singleton*, se hará una pausa de tres segundos y se intentará crear una nueva instancia (llamando a la propiedad *Instance*, ya que recordemos que el constructor es privado). Si se trata de la misma instancia, se mostrará exactamente la misma hora: **Ver código del Main del proyecto PatronSingleton**



Uso de un Singleton

Por norma general, una instancia *Singleton* no se utilizará de esta manera, sino que **se usará como si fuera un simple atributo estático** (de hecho, es lo que es). Por lo tanto, en lugar de invocar el método *Singleton.Instance*, lo que haremos será utilizarlo como si la instancia siempre hubiese existido: recordemos que si el atributo privado *_instancia* no tiene valor, el propio *getter* de la *Property Instance* se encargará de invocar al constructor. Por lo tanto, este elemento se utilizará como si de un objeto cualquiera se tratase.

Pero… el código tiene un problema: **no es seguro en entornos multi-hilo** (*multithreading*)… Si dos hilos de ejecución entran en la propiedad *Instance* al mismo tiempo, **es posible que se creen dos instancias** de nuestra clase *Singleton*.

Existen varias posibilidades para resolver este problema. La primera de ellas se basa en la llamada **inicialización estática**, que es viable en entornos *.NET* pero que no es funcional en otros lenguajes.

Inicialización estática

La inicialización estática se basa en confiar en que el propio *Framework* instancie la clase la primera vez que es accedida, independientemente del hilo que ejecute el código. Para ello necesitaremos hacer que nuestra clase cumpla con dos preceptos:

- Declarar nuestra clase como *sealed* (clase sellada*), de modo que no sea posible crear clases heredadas que puedan crear nuevas instancias.
- Declarar el atributo interno *_instancia* como *readonly*, de modo que únicamente pueda ser modificado en la inicialización o en el constructor.
- Llamar al constructor en la propia declaración del atributo, de modo que **sea instanciado automáticamente la primera vez que se haga uso de él**.

  Ver clase SingletonEstatico.cs del proyecto PatronSingletonStatic, el código del Main es igual al del proyecto anterior.

En el momento en el que se acceda por primera vez a *SingletonEstatico.Instance*, el atributo *_instancia* será instanciado por el constructor, llamado por el propio *Framework*. Dado su carácter de sólo lectura, esto sólo podrá realizarse una única vez. Esta operación es *thread-safe*, por lo que se podrá utilizar en entornos multi-hilo sin que se presenten problemas de concurrencia. Esta forma de instanciar objetos se conoce por el nombre de *Lazy Instantiation* o **instanciación diferida**.

Nota*: Una clase con el modificador sellado, evita que otras clases hereden de ella. En el ejemplo siguiente, la clase `B` hereda de la clase `A`, pero ninguna clase puede heredar de la clase `B`:

```
class A {}

sealed class B : A {}
```

https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/sealed

¿Cuándo utilizar este patrón? Ejemplos reales

La existencia de este patrón es relativamente clara: debe utilizarse cuando nuestra aplicación precise que una clase se instancie una única vez. **Siempre que exista un objeto de carácter global**, **que apenas cambie con el tiempo**, será susceptible de ser diseñado mediante un patrón *Singleton*.

Ejemplos reales de utilización de patrones *Singleton* podrían ser:

- Utilitarios u objetos que hagan *Logging*, logueo de información dentro de las aplicaciones: **no nos interesa que más de una instancia escriba en un log a la vez**.
- Utilitarios de configuración del sistema, así como clases encargadas de realizar el cacheo de memoria o el balanceo de carga.
- El tener un objeto que mantenga una única conexión abierta o acceso exclusivo a un archivo, o base de datos, o a otro recurso.