MOTIVACIÓN

En muchas aplicaciones se observa que es frecuente que desde varios clientes distintos se precise referenciar a un mismo elemento. A la vista de esta realidad, sería conveniente disponer de algún mecanismo que garantice que no hay más de una instancia de ese elemento y que, por tanto, todos las clases que lo referencian están viendo el mismo objeto.

OBJETIVO

Garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

DESCRIPCIÓN

El patrón de diseño **singleton** (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase. Su finalidad consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón *singleton* se implementa definiendo en la clase un método que crea una instancia del objeto sólo si todavía no existe ninguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor definiéndolo privado.

Ejemplos habituales de aplicación de este patrón serían aquellos en los que una clase debe controlar el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto conjunto de datos debe estar accesible para todos los demás objetos de una aplicación.

El patrón *Singleton* provee una única instancia global gracias a que:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase (estático).
- Declara el constructor de clase como privado para que no sea instanciable directamente.

DOS POSIBLES IMPLEMENTACIONES

1. ***Inicialización bajo demanda:***

```
public class Singleton {
        private static Singleton instanciaUnica = new Singleton();

        // El constructor privado no permite que otras clases generen más objetos de este tipo
        private Singleton() {}

        // Método de acceso que permite que otras clases accedan al objeto único de este tipo
        public static Singleton getInstance() {
                return instanciaUnica;
        }
}
```

2. **Inicialización diferida:**

```
public class Singleton {
        private static Singleton instanciaUnica = null;

        // El constructor privado no permite que otras clases generen más objetos de este tipo
        private Singleton() {}

        // Método de acceso que crea la instancia, si todavía no existe y permite que otras
        // clases accedan al objeto único de este tipo
        public static Singleton getInstance() {
```

```
                if (instanciaUnica == null) {
                        instanciaUnica = new Singleton();
                }
                return instanciaUnica;
        }
}
```

NOTA:
Para asegurar que se cumpla el requerimiento de "única instancia" del singleton; la clase debería producir un objeto no clonable. Entonces, se debería impedir la clonación sobrescribiendo el método "clone" de la siguiente manera:

```
public Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
}
```

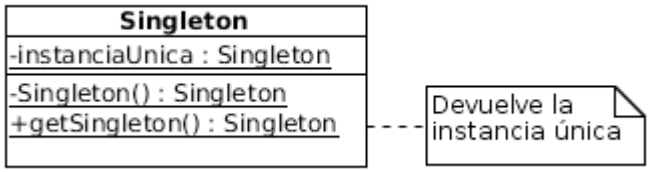

Figura 1: Diagrama UML del patrón Singleton,

UTILIZACIÓN

Usar el patrón singleton cuando:

- Deba haber exactamente una instancia de una clase y ésta deba ser accesible a los clientes desde un punto de acceso conocido.
- La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código.

AVISO!!

Dado que el Singleton es un patrón muy sencillo, en ocasiones es sobreutilizado y muchas veces en forma incorrecta.

¿Esto quiere decir que los Singleton son malos y no hay que usarlos? Definitivamente NO. Pero como todo, debe utilizarse en su justa medida y en el contexto adecuado.

VENTAJAS

- Acceso controlado a la única instancia. Puede tener un control estricto sobre cómo y cuándo acceden los clientes a la instancia.
- Espacio de nombres reducido. El patrón Singleton es una mejora sobre las variables globales.
- Permite el refinamiento de operaciones y la representación. Se puede crear una subclase de Singleton.

Ejemplo "No Software"

La oficina del Lehendakari es un Singleton. El estatuto vasco especifica el medio para elegir un Lehendakari, limita el término “oficina del Lehendakari” y define el orden de sucesión. Por tanto, puede haber como mucho un Lehendakari activo en un momento dado. Sin importar la identidad personal del Lehendakari activo, el título de “Lehendakari” es un punto global de acceso que identifica a la persona que está en la “oficina del Lehendakari”. Es posible encontrar a diferentes personas que hayan sido Lehendakari, pero para todos los casos, “la oficina del Lehendakari” fue la misma.

Ejemplo Software

Definir un contador usando el patrón singleton.

```
public class Contador {
        private static Singleton unContador = null;
        private int valor = 0;

        // El constructor privado impide que otras clases generen más objetos Contador
        private Contador() {}

        // Método de acceso crea la instancia, si todavía no existe, y permite que otras clases
        // accedan al objeto único de este tipo
        public static Contador getInstance() {
                if (unContador == null) {
                        unContador = new Contador();
                }
                return unContador;
        }

        public static int incrementarContador() {
                valor = valor+1;
                return valor;
        }
}
```