



**TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA**

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Tarea
Patrón singleton

Docente
José de Jesús Parra Galaviz

SEMESTRE agosto – diciembre 2021

**INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y
COMUNICACIONES**

Patrones de diseño
Sergio Alberto Garza Aguilar 15211700

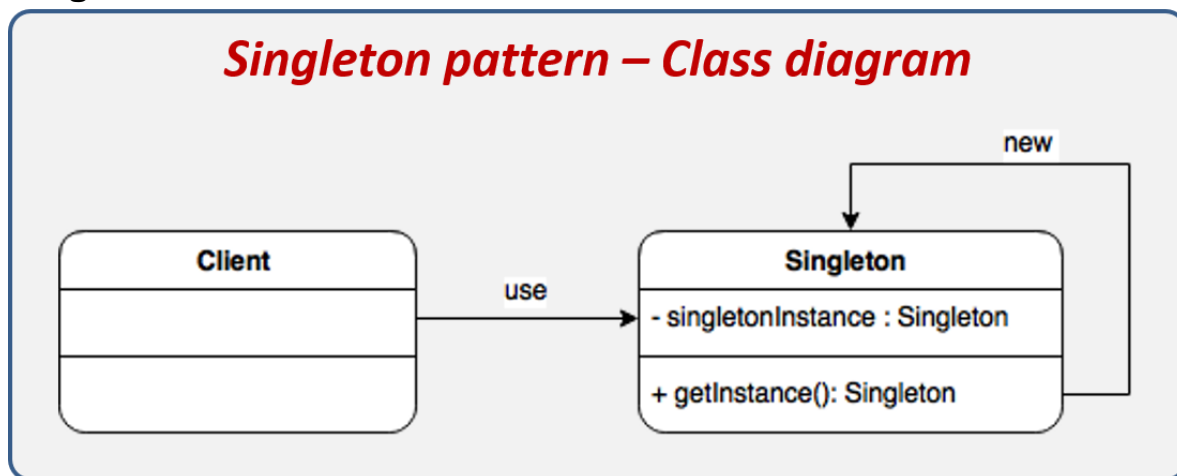
15 de octubre del 2021

Descripción

El patrón de diseño Singleton (soltero) recibe su nombre debido a que sólo se puede tener una única instancia para toda la aplicación de una determinada clase, esto se logra restringiendo la libre creación de instancias de esta clase mediante el operador new e imponiendo un constructor privado y un método estático para poder obtener la instancia. 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.

Singleton es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

Diagrama



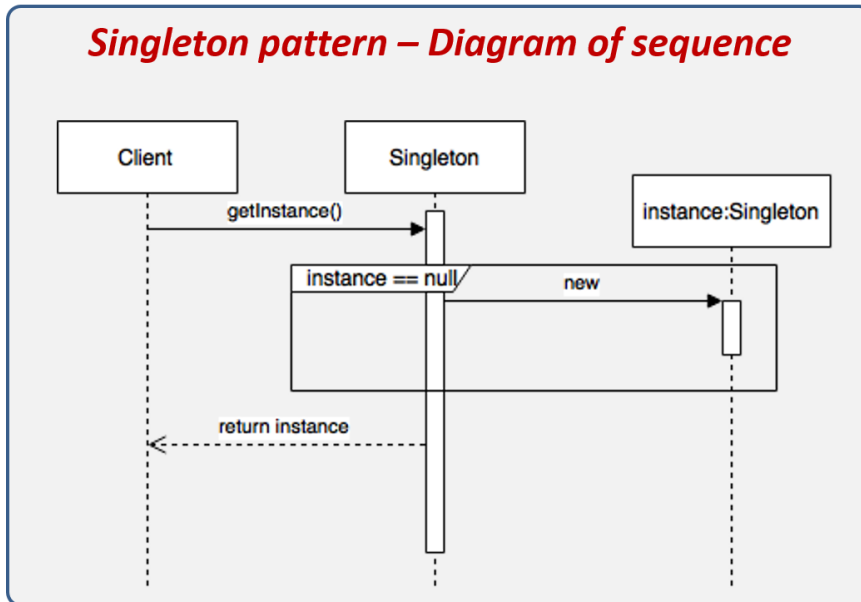
Componentes que conforman el patrón son los siguientes:

- **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.

Implementación

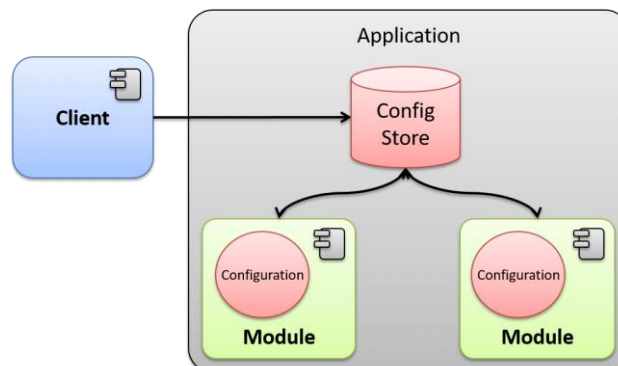
1. Añade un campo estático privado a la clase para almacenar la instancia Singleton.
2. Declara un método de creación estático público para obtener la instancia Singleton.
3. Implementa una inicialización diferida dentro del método estático. Debe crear un nuevo objeto en su primera llamada y colocarlo dentro del campo estático. El método deberá devolver siempre esa instancia en todas las llamadas siguientes.
4. Declara el constructor de clase como privado. El método estático de la clase seguirá siendo capaz de invocar al constructor, pero no a los otros objetos.
5. Repasa el código cliente y sustituye todas las llamadas directas al constructor de la instancia Singleton por llamadas a su método de creación estático.

Diagrama de secuencia



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

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, cargará la configuración inicial y está disponible para toda la aplicación.



Ventajas

- Puedes tener la certeza de que una clase tiene una única instancia.
- Obtienes un punto de acceso global a dicha instancia.
- El objeto Singleton solo se inicializa cuando se requiere por primera vez.

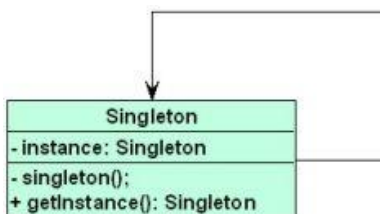
Desventajas

- Vulnera el Principio de responsabilidad única. El patrón resuelve dos problemas al mismo tiempo.
- El patrón Singleton puede enmascarar un mal diseño, por ejemplo, cuando los componentes del programa saben demasiado los unos sobre los otros.
- El patrón requiere de un tratamiento especial en un entorno con múltiples hilos de ejecución, para que varios hilos no creen un objeto Singleton varias veces.
- Puede resultar complicado realizar la prueba unitaria del código cliente del Singleton porque muchos frameworks de prueba dependen de la herencia a la hora de crear objetos simulados (mock objects). Debido a que la clase Singleton es privada y en la mayoría de los lenguajes resulta imposible sobrescribir métodos estáticos, tendrás que pensar en una manera original de simular el Singleton. O, simplemente, no escribas las pruebas. O no utilices el patrón Singleton.

Ejemplo de código

Utilizaremos el patrón Singleton cuando por alguna razón necesitemos que exista sólo una instancia (un objeto) de una determinada Clase.

Dicha clase se creará de forma que tenga una propiedad estática y un constructor privado, así como un método público estático que será el encargado de crear la instancia (cuando no exista) y guardar una referencia a la misma en la propiedad estática (devolviendo ésta).



En el siguiente código en Java se intenta obtener cinco veces una instancia de la clase Coche:

Main.java

```

01. package Singleton;
02.
03. public class Main
04. {
05.     public static void main(String[] args)
06.     {
07.         for(int num=1; num<=5; num++)
08.         {
09.             Coche.getInstance();
10.         }
11.     }
12. }

```

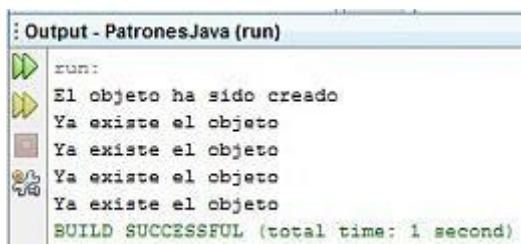
Coche.java

```

01. package Singleton;
02.
03. public class Coche
04. {
05.     private static Coche instancia;
06.
07.     // -----
08.
09.     private Coche() {
10.     }
11.
12.     // -----
13.
14.     public static Coche getInstance()
15.     {
16.         if (instancia == null) {
17.             instancia = new Coche();
18.             System.out.println("El objeto ha sido creado");
19.         }
20.         else {
21.             System.out.println("Ya existe el objeto");
22.         }
23.
24.         return instancia;
25.     }
26. }

```

Salida



```

Output - PatronesJava (run)
run:
El objeto ha sido creado
Ya existe el objeto
Ya existe el objeto
Ya existe el objeto
Ya existe el objeto
BUILD SUCCESSFUL (total time: 1 second)

```

Fuentes de consulta.

- Alexander Shvets (2014-2021) Sumerjete en los patrones de diseño, Singleton.
(<https://refactoring.guru/es/design-patterns/singleton>)
- (2020) Informaticapc, Patrones de diseño de software, patron singleton
(<https://informaticapc.com/patrones-de-diseno/singleton.php>)
- Oscar Blancarte (08 marzo del 2021), introducción a los patrones de diseño un enfoque practico, Singleton Patter,
(<https://reactiveprogramming.io/blog/es/patrones-de-diseno/singleton>)