SINGLETON EN MULTI-HILOS

JHON EDDI MALAGÓN – 20151020021

Resumen: En este artículo se realizará un pequeño resumen del patrón creacional singleton, para lograr profundizar más cuando el patrón utiliza más de un hilo.

Introducción: El patrón singleton o también conocido como instancia única es utilizado en la ingeniería de software para la creación de software. Este permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Tiene como objetivo el garantizar que una clase tenga solo una única instancia.

Singleton en multi-hilos

El principio de singleton es suficiente en la implementación de un ambiente de un solo hilo (single-thread), pero en un ambiente compartido, es decir de múltiples hilos (multi-thread) tiene serios problemas debido a cuestiones de sincronización y concurrencia, puede crearse más de una instancia del miembro instance, imaginemos que dos hilos evalúan la condición instance == null y en ambos casos es verdadera. En este caso ambos hilos crearán la instancia, violando el propósito del patrón singleton.

- Si el singleton es absolutamente sateless (es decir, no mantiene ningún tipo de estado) puede no ser un problema.
- Si usamos C++, se puede producir un memory leak, dado que sólo se va a eliminar uno de los objetos aunque hayamos creado dos.
- Si el singleton es statefull (mantiene estado) se pueden producir errores sutiles. Por ejemplo, si se modifca el estado del objeto en el constructor, pueden producirse inconsistencias, dado ese código de inicialización se ejecuta más de una vez.

Los problemas que se producen a raíza de esto pueden ser muy difíciles de detectar. La creación dual suele producirse en forma intermitente e incluso puede no suceder (no es determinista).

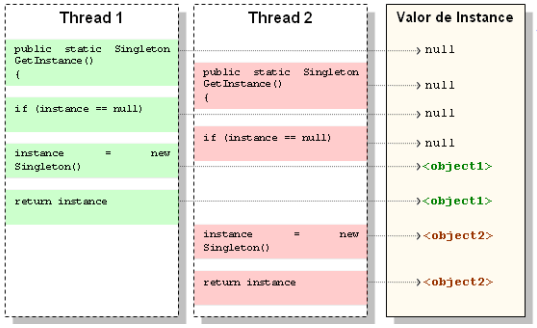

Figura 1: Representación de los problemas en la implementación del singleton.

Para solucionar este problema se puede recurrir a:

Sentencia Lock

Esta se encarga de realizar las siguientes tareas:

1. Obtiene un bloqueo exclusivo (mutual-exclusive lock) para un objeto.
2. Ejecuta un conjunto de sentencias.
3. Luego suelta el bloqueo.

Double-Check Locking

Este es un idioma ampliamente cifrado y eficiente para implementar inicialización tardia (lazy inicialization) en entornos multihilo:

- Se evitan bloqueos innecesarios envolviendo la llamada al new con otro testeo condicional.
- Soporte para ambientes multihilo.

Este es mejor que la sentencia lock, debido a que el bloqueo solo es necesario cuando se crea la instancia de instance. De esta forma, al incurrir en menos bloqueos, obtenemos un mejor rendimiento.

Conclusión

El principio creacional singleton puede llegar a ser una herramienta de doble filo, ya que cubre las necesidades de instanciar un solo objeto con gran facilidad, pero a cambio nos genera problemas al momento de crearse más de una instancia, para ello recurrimos a la sentencia lock o al Double-Check Locking también, que nos limita a través de bloqueos para restringir la creación de instancias.