Tarea No. 11. Singleton

Barrera Pérez Carlos Tonatihu
Profesor: José Asunción Enríquez Zárate
Web Application Development
Grupo: 3CM9

12 de marzo de 2019

Índice

1. Introducción

El patrón singleton es un patrón de diseño que restringe las instancias de una clase a un objeto. Existen diferentes formas de implementarlo, para esto es necesario tener conocimiento sobre variables de clase estáticas y modificadores de acceso.

2. Desarrollo

En programación orientada a objetos una clase singleton es una clase que tiene solo un objeto a la vez. Después de la primera inicialización si se intenta inicializar de nuevo, la nueva variable también apuntara a la primera instancia creada. Así que cualquier modificación que se haga a cualquier variable dentro de la clase a través de la instancia, afectara la variable de la instancia única.

Para diseñar una clase singleton es necesario:

- Hacer el constructor privado.
- Escribir un método estático que retorne el tipo de objeto de la clase singleton. Aquí se aplica el concepto de inicialización perezosa.

2.1. Implementación clásica

Esta es la implementación más conocida, solo se tiene una instancia estatica de la clase cuando se crea, sin embargo, presenta el grave problema de que no es seguro en hilos. Ya que de existir dos o más hilos se podrian crear dos objetos del singleton.

```
1  class Singleton {
2      private static Singleton obj;
3
4      /*
5      * Constructor privado para forzar a utilizar el metodo
6      * getInstance() para crear el objeto Singleton
7      */
8      private Singleton() {}
9
10     public static Singleton getInstance() {
11         if (obj==null)
12             obj = new Singleton();
```

```
13          return obj;
14      }
15  }
```

2.2. Implementación de hilo sincronizado

Esta implementación es segura ya que se utiliza el modificador synchronized, por lo que solo un hilo puede ejecutar el método a la vez. La desventaja es que esta implementación es bastante costosa en cuanto al desempeño; pero si este no es un problema es una buena solución.

```
1  class Singleton {
2      private static Singleton obj;
3
4      private Singleton() {}
5
6      // Solo un hilo puede ejecutar este metodo a la vez
7      public static synchronized Singleton getInstance() {
8          if (obj==null)
9              obj = new Singleton();
10         return obj;
11     }
12 }
```

2.3. Inicialización impaciente

En esta opción, se inicializa la variable al cargar la clase, la JVM ejecuta el inicializador estático y por lo tanto es seguro en hilos. Se usa este método cuando la clase singleton es ligera y es usado a lo largo de la ejecución del programa.

```
1  class Singleton {
2      private static Singleton obj = new Singleton();
3
4      private Singleton() {}
5
6      public static Singleton getInstance() {
7          return obj;
8      }
9  }
```

2.4. Double checked locking

Una vez qu un objeto es creado, la sincronización deja de ser útil debido a que ahora el objeto no sera nulo y cualquier secuencia de operaciones conducirá a resultados consistentes.

Por lo que solo tendremos bloqueo en el método getInstance() una sola vez, cuando el objeto es nulo. En esta forma solo se sincroniza una sola vez.

Por ultimo, el objeto se declara volatil, lo cual asegura que múltiples hilos brinden la variable objeto correctamente cuando es inicializado.

```
1  class Singleton {
2      private volatile static Singleton obj;
3
4      private Singleton() {}
5
6      public static Singleton getInstance() {
7          if (obj == null) {
8              // Para hacerlo el hilo seguro
9              synchronized (Singleton.class) {
10                 /*
11                 * Se vuelve a checar ya que varios hilos
12                 * pueden alcanzar el paso anterior
13                 */
14                 if (obj==null)
15                     obj = new Singleton();
16             }
17         }
18         return obj;
19     }
20 }
```

3. Conclusiones

Es conocer los requerimientos de la aplicación que se este desarrollando es importante, ya que como se muestra en esta tarea existen diversas formas de implementar este patrón, así resulta bastante útil el saber cual es el que más se ajusta a nuestras necesidades.