2. EXPLICACIÓN DETALLADA DE LOS PATRONES DE DISEÑO UTILIZADOS EN LOS EJEMPLOS

2.1. SINGLETON [GoF95]

2.1.1. Objetivo

Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia.

2.1.2. Aplicabilidad

El patrón Singleton se puede utilizar en los siguientes casos:

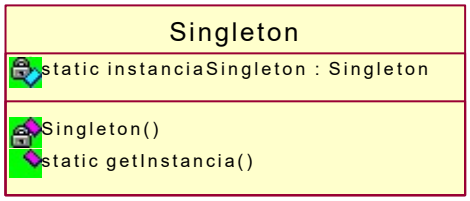
- Debe haber exactamente una instancia de la clase.
- La única instancia de la clase debe ser accesible para todos los clientes de esa clase.

2.1.3. Solución

El patrón Singleton es relativamente simple, ya que sólo involucra una clase.

Una clase Singleton tiene una variable static que se refiere a la única instancia de la clase que quieres usar. Esta instancia es creada cuando la clase es cargada en memoria. Tú deberías de implementar la clase de tal forma que se prevenga que otras clases puedan crear instancias adicionales de una clase Singleton. Esto significa que debes asegurarte de que todos los constructores de la clase son privados.

Para acceder a la única instancia de una clase Singleton, la clase proporciona un método static, normalmente llamado *getInstancia*, el cual retorna una referencia a la única instancia de la clase.



Clase Singleton.

2.1.4. Consecuencias

- Existe exactamente una instancia de una clase Singleton.
- Otras clases que quieran una referencia a la única instancia de la clase Singleton conseguirán esa instancia llamando al método static *getInstancia* de la clase. Controla el acceso a la única instancia.
- Tener subclases de una clase Singleton es complicado y resultan clases imperfectamente encapsuladas. Para hacer subclases de una clase Singleton, deberías tener un constructor que no sea privado. También, si quieres definir una subclase de una clase Singleton que sea también Singleton, querrás que la subclase sobrescriba el método *getInstancia* de la clase Singleton. Esto no será posible, ya que métodos como *getInstancia* deben ser static. Java no permite sobrescribir los métodos static.

2.1.5. Implementación

Para forzar el carácter de una clase Singleton, debes codificar la clase de tal forma que prevengas que otras clases creen directamente instancias de la clase. El modo de realizarlo es declarar todos los constructores de la clase privados. Tener cuidado de declarar al menos un constructor privado. Si una clase no declara ningún constructor, entonces un constructor público es automáticamente generado por ella.

Una variación común del patrón Singleton ocurre en situaciones donde la instancia de un Singleton podría no ser necesaria. En estas situaciones, puedes posponer la creación de la instancia hasta la primera llamada a *getInstancia*.

Otra variación sobre el patrón Singleton desciende del hecho de que la política de instanciación de una clase está encapsulada en el método de la clase *getInstancia*. Debido a esto, es posible variar la política de creación. Una posible política es tener el método *getInstancia* que retorne alternativamente una de las dos intancias o crear periódicamente una nueva instancia para ser retornada por *getInstancia*. Es decir se puede permitir un número variable de instancias.

2.1.6. Usos en el API de Java

En el API de Java la clase *java.lang.Runtime* es una clase Singleton, tiene exactamente

una única instancia de la clase. No tiene constructores públicos. Para conseguir una referencia a su única instancia, otras clases deben llamar a su método static *getRuntime*.

2.1.7. Patrones relacionados

Puedes utilizar el patrón Singleton con muchos otros patrones. En particular, es a menudo utilizado con los patrones Abstract Factory, Builder, y Prototype.

El patrón Singleton tiene alguna similitud con el patrón Cache Management. Un Singleton es funcionalmente similar a un Cache que contiene solamente un objeto.