|  |
| --- |
| DEFOIS, FARISSE, HAMANI, LOPPY, VERGINE |
| Documentation D’un Design Pattern |
| Singleton |

|  |
| --- |
| Soufiane HAMANI  26/10/2017 |

Table des matières

[Nom du Pattern 2](#_Toc497743707)

[Description du Pattern 2](#_Toc497743708)

[Exemple de Diagramme UML (Problème et Solution) 3](#_Toc497743709)

[Les Domaines d’Application du Pattern 4](#_Toc497743710)

[Exemple de Code en C# 4](#_Toc497743711)

# Nom du Pattern

SINGLETON:

Le singleton permet de s'assurer qu'une seule instance d'un objet donné sera instanciée pendant toute la durée de votre application. Une seule dans l'espace comme dans le temps, c'est-à-dire :

l'espace représenté par la mémoire - vous êtes certain de l'unicité de l'instance à un moment donné

le temps - vous vous assurer de l'unicité de l'instance à chaque appel. J'entends par la que vous êtes certain que c'est la même instance que vous référer quel que soit le temps écoulé entre deux appels.

# Description du Pattern

L’objectif est d’ajouter un contrôle sur le nombre d’instances que peut retourner une classe.

La première étape consiste à empêcher les développeurs d’utiliser le ou les constructeur(s) de la classe pour l’instancier. Pour cela il suffit de déclarer privé tous les constructeurs de la classe. Attention dans certains langages une classe sans constructeur possède un constructeur implicite par défaut (c’est notamment le cas de Java). Il faut donc que celui-ci soit déclaré explicitement en privé.

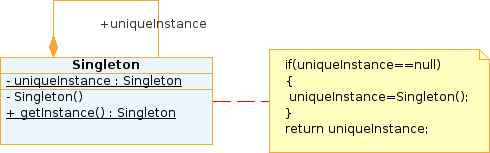
Une fois cette étape accomplie, il est possible d’instancier cette classe uniquement depuis elle même, ce qui n’a pas beaucoup de sens. Comment allons nous faire pour permettre aux développeurs de l’utiliser ?

Nous allons construire un pseudo constructeur. Pour cela il faut déclarer une méthode statique qui retournera un objet correspondant au type de la classe. L’avantage de cette méthode par rapport à un constructeur, est que l’on peut contrôler la valeur que l’on va retourner. Le fait que cette méthode soit déclarée statique permet de l’appeler sans posséder d’instance de cette classe. A noter que, par convention, ce pseudo constructeur est nommé getInstance.

Pour en finir avec le concept de base du Singleton voyons comment implémenter cette méthode.

Tout d’abord il faut créer un attribut statique qui va permettre de stocker l’unique instance de la classe. Ensuite, dans le pseudo constructeur on va tester cet attribut. Si celui-ci est nul alors on crée une instance de la classe et on stocke sa valeur dans cet attribut. Sinon c’est que l’attribut possède déjà une instance de la classe. Dans tous les cas la méthode retourne la valeur de l’attribut possédant l’unique instance de la classe.

# Exemple de Diagramme UML



# Les Domaines d’Application du Pattern

Le Singleton répond à deux exigences :

* Garantir qu'une unique instance d'une classe donné sera créée
* Offrir un point d'accès universel à cette instance.

Ce design pattern est tout indiqué pour implémenter des services qui :

* sont fonctionnellement uniques au sein de l'application (ex: système de logging centralisé, gestion de la configuration...)
* doivent pouvoir être appelés par toutes les couches de l'application. Il serait en effet peu pratique de passer une référence au service à toutes les classes devant l'utiliser.

# Exemple de Code en C#

Ce code représente une Implémentation

|  |
| --- |
| **public** **class** **Singleton**  {  **private** **static** Singleton \_instance;  **static** **readonly** object instanceLock = **new** object();  **private** Singleton()  {    }  **public** **static** Singleton getInstance()  {  **lock** (instanceLock)  {  **if** (\_instance == **null**)  \_instance = **new** Singleton();      **return** \_instance;  }  }  } |