1. Le design pattern Singleton

Pour la résolution de notre solution logicielle multicouches, nous allons nous initier aux patrons de conception ou design pattern.

Ces patrons de conception tentent de répondre à des problèmes de conception de logiciels dans le cadre de la programmation orientée objet.

Ces modèles résultent de travaux de différents groupes de développeurs et ont été éprouvés par de nombreuses implémentations.

Ils peuvent être catalogués en fonction de leur couche de spécialisation (interface, métier, accès aux données) ou en fonction de leur vocation.

Ce dernier point correspond à la classification retenue par le Gof (Gang of Four) qui fut à l'origine de la formalisation d'une vingtaine de patrons de conception :

- Pattern de construction : ils ont pour but la création d'objets
- Patterns de structuration : ils favorisent l'organisation de la hiérarchie des classes et de leurs relations.
- Patterns de comportement : ils proposent des solutions pour organiser les interactions et répartir les traitements entre les objets.

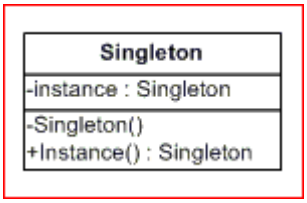
Nous allons ici découvrir un pattern de construction, le pattern singleton.

Le pattern singleton permet de s'assurer qu'il n'existe qu'une seule instance d'un type en mémoire.

Pas très éloigné de la notion de type statique, le singleton a pour avantages par rapport au modèle statique :

- de permettre l'héritage.
- de ne pas consommer de ressources s'il n'est pas instancié.

Voici le diagramme UML de celui-ci.



Un champ privé qui figure l'instance unique du singleton ainsi qu'un constructeur « privatisé ».
Pour privatiser le constructeur par défaut, il faut tout d'abord le définir et préciser une accessibilité privée.

Une méthode (ou une propriété publique en lecture seule) qui retourne l'instance unique du singleton.

Nous ajouterons aussi un mécanisme qui permet de s'assurer que l'opération de lecture de la variable statique stockant l'instance ne peut pas faire l'objet d'un accès concurrentiel.

Nous utilisons pour cela le verrouillage d'un objet le temps du traitement.

Ce qui donne l'exemple d'implémentation suivant :

```
public class Singleton
{
    private static Singleton _instance = null;

    // Ajout d'un objet pour s'assurer du verrouillage de code
    // Les threads ne sont pas isolés et safe
    private static object _verrou = new object();

    private Singleton() { }

    public static Singleton Instance
    {
        get
        {
            lock (_verrou)
            {
                if (_instance == null)
                {
                    _instance = new Singleton();
                }
            }
            return _instance;
        }
    }
}
```