

### TD-TP : Le Design Pattern de Création « Singleton »

**Singleton** : Garantit qu'une classe n'a qu'une seule instance, il fournit à celle-ci un point d'accès de type global.

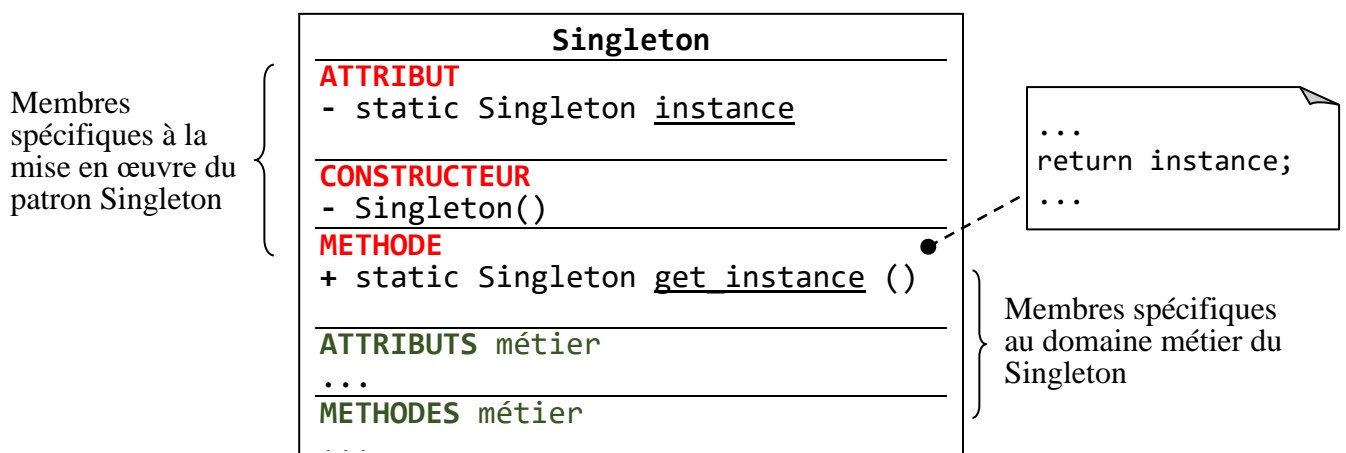
**Motivation** : Il est important pour certains systèmes de n'avoir qu'une seule instance. Alors qu'il peut y avoir plusieurs imprimantes dans un système, il ne doit y avoir qu'un seul serveur d'impression. De même qu'il ne doit y avoir qu'un seul gestionnaire de fenêtres, une comptabilité n'est que pour une seule entreprise.

**Indications d'utilisation** : Il ne doit y avoir exactement qu'une seule instance d'une classe, qui de plus doit être accessible aux clients par un point bien déterminé.

**Structure** : Le constructeur Singleton() est `private` ce qui empêche la création d'instances par n'importe quel client ayant connaissance de l'existence de la classe Singleton().

Pour remédier à l'interdiction d'accès au constructeur (cf. `private`), la classe Singleton définit une méthode `public` nommée `get_instance()`. C'est une méthode de classe (cf. `static`) qui donne accès aux clients, à son unique instance. L'adresse de son unique instance est stockée dans l'attribut privé `instance`.

La méthode de classe `get_instance()` doit donc être invoquée via la classe Singleton, cad avec la syntaxe : `Singleton.get_instance()`;



### Schéma de mise en œuvre

```
// Programme principal illustrant l'usage du patron Singleton
public static void main(String[] args){
    // Pointeurs capables de pointer sur une instance de la classe Singleton
    Singleton ptr1, ptr2;

    // Initialisation des deux pointeurs (la première crée l'instance)
    ptr1 = Singleton.get_instance(); // Possible car méthode statique
    ptr2 = Singleton.get_instance(); // Possible car méthode statique

    // On accède aux membres (méthodes et attributs) du singleton
    // indifféremment par ptr1 et/ou par ptr2
    ...
}
```

```
/// DEFINITION D'UNE CLASSE Singleton
public class Singleton {
    /// Membres spécifiques à la mise en œuvre du mécanisme du design pattern Singleton
    // ATTRIBUT
    // Référence privée, et au niveau de la classe, de l'unique instance
    private static Singleton instance = null;

    // CONSTRUCTEUR
    private Singleton (aaa) {bbb} // Privé avec d'éventuels paramètres

    // METHODE d'obtention de l'instance qui se charge, d'appeller le constructeur
    public static Singleton get_instance(ccc) {
        if (instance == null) { // Equivaut à : if (Singleton.instance == null)
            // Invoque le constructeur avec d'éventuels paramètres
            instance = new Singleton(ddd);
            ...
        }
        return instance;
    }

    /// Membres spécifiques à la mise en œuvre métier de l'unique instance
    // (ils n'ont pas à être static, les attributs appartiennent à l'objet)
    // ATTRIBUTS métier
    // ...

    // METHODES métier
    // ...
}
```

### Travail à faire

Il s'agit de créer un projet Singleton qui met en œuvre le design pattern Singleton. L'objectif est de créer et manipuler une classe SingletonChefCuisinier dont ne peut exister qu'une seule instance. Les mécanismes du Design Pattern Singleton seront donc mobilisés.

1. Donner le code de la classe SingletonChefCuisinier. Outre les membres à déclarer en **static**, conformément au patron Singleton, l'unique instance de la classe comportera les attributs métier privés **nom** et **prénom** de l'unique chef cuisinier, attributs qui seront accessibles par les typiques méthodes publiques **get**&**set**.
2. Dans un **main()** d'une classe **TesterSingleton**, mettre en œuvre et tester le fonctionnement du design pattern Singleton appliqué à la classe **SingletonChefCuisinier**. Pour cela, le **main()** :
  - a) Déclare **leCuisinier** et **leChef**, deux pointeurs d'objet de type **SingletonChefCuisinier**
  - b) Via **get\_instance()** affecte à **leCuisinier** l'adresse de l'unique instance de la classe **SingletonChefCuisinier**
  - c) Via le pointeur **leCuisinier** affecte à cette unique instance le **nom** de «Navarro»
  - d) Via **get\_instance()** affecte à **leChef** l'adresse de l'unique instance de la classe **SingletonChefCuisinier**
  - e) Affiche le **nom** de l'instance pointée par **leChef** et constater qu'il s'agit bien du même et unique : l'excellent chef «Navarro», the king of the pot agé !!!
3. Réflexions sur la méthode **get\_instance()**
  - f) Pensez-vous que techniquement, la méthode **get\_instance()** pourrait admettre des paramètres pour initialiser les attributs **nom** et **prénom** et déléguer au constructeur la responsabilité d'initialiser les attributs de l'instance ? Si oui quelles seraient les modifications à apporter ?
  - g) Pensez-vous que la solution technique à la question f. soit conceptuellement saine ?