UNIVERSITÉ CHEIKH ANTA DIOP DE DAKAR

ECOLE SUPÉRIEURE POLYTECHNIQUE

DÉPARTEMENT GÉNIE INFORMATIQUE

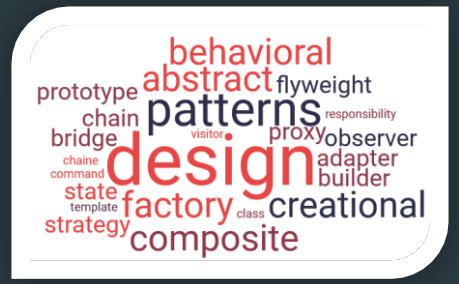
PATRONS DE CONCEPTION

COURS INTRODUCTIF

Formateur

Dr Mouhamed DIOP

mouhamed.diop@esp.sn



Le Pattern Singleton

Contexte

- De nombreux objets doivent exister en un seul exemplaire
  - Pools de thread
  - Cache
  - Serveur
  - Gestionnaire des paramètres d'un registre
  - Gestion des pilotes
  - ...
- Une instanciation multiple entrainerait des incohérences

Instanciation des objets

- Utilisation du mot clé new : MaClasse classe = new MaClasse()
  - Il devient alors possible d'avoir plusieurs instances de MaClasse
- Rendre privé le constructeur de MaClasse

```
public class MaClasse
{
    private MaClasse()
    {
        ....
    }
}
```

  - Il devient impossible d'instancier la classe
- Comment faire alors pour ne pouvoir obtenir qu'une seule instance de la classe ?

Mécanisme d'instanciation d'objet Singleton

- Rendre privé le constructeur de la classe
  - Pour rendre impossible l'instanciation de la classe depuis l'extérieur
- Fournir une méthode de classe chargée de la création et du partage de l'objet

```
public class MaClasse
{
    public static MaClasse getInstance()
    {
        return new MaClasse();
    }
}
```

- S'assurer qu'une seule instance soit créée par getInstance

Le pattern Singleton

- Le pattern singleton est constitué d'une seule classe
  - Convention qui garantit qu'un seul objet est instancié pour une classe donnée
  - Fournit une méthode de classe unique retournant cette instance
- Garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance
- Un singleton est une classe normale
  - Elle peut donc possèder d'autres variables d'instances et méthodes

Le pattern Singleton : implémentation

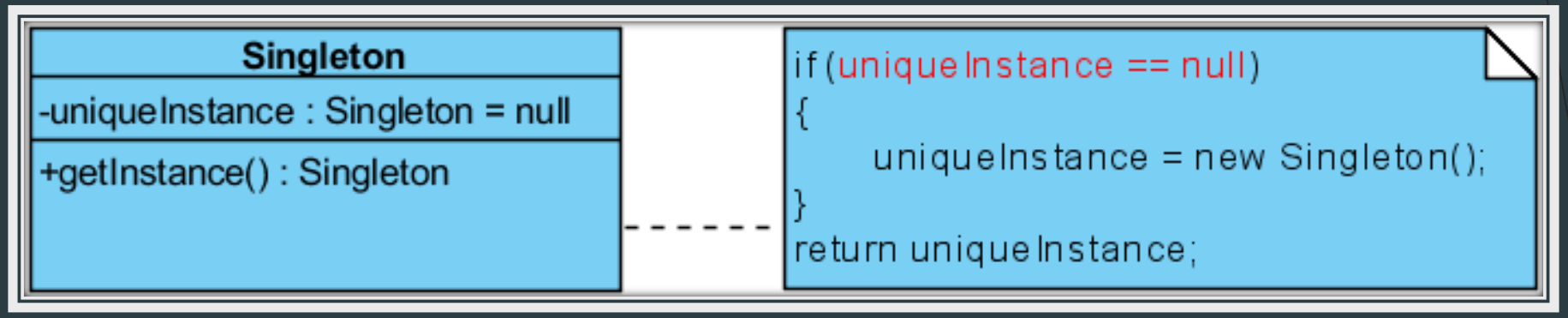
```
public class Singleton
{
    private static Singleton uniqueInstance;

    private Singleton() {}

    public static Singleton getInstance()
    {
        if (uniqueInstance == null)
        {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
}
```

Le pattern Singleton

- Diagramme de classe



- Le seul participant est la classe Singleton offrant l'accès unique par sa méthode de classe getInstance
- Chaque client accède à l'unique instance par la méthode getInstance
  - L'instanciation par l'opérateur new étant bloquée

Le pattern Singleton

- Conditions d'utilisation
  - Il ne doit y avoir qu'une seule instance de la classe
  - L'instance de la classe doit être accessible à d'autres classes
    - L'accès leur est garanti via la méthode de classe

Le pattern Singleton et le multithreading

- Que se passe-t-il si deux threads accèdent simultanément à la méthode getInstance ?
  - Risque de création de plusieurs instances
- Solution
  - Obliger chaque thread à attendre son tour avant d'entrer dans la méthode getInstance
  - Rendre l'exécution de la méthode d'instanciation mutuellement exclusive
    - Deux threads ne peuvent l'utiliser en même temps

Le pattern Singleton et le multithreading

- Utilisation du mot clé synchronized en Java

```
public class Singleton
{
    private static Singleton uniqueInstance;

    private Singleton() {}

    public static synchronized Singleton getInstance()
    {
        if (uniqueInstance == null)
        {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
}
```

Le pattern Singleton et le multithreading

- Inconvénients du mot clé synchronized
  - Il n'est utile que lors du premier passage
    - Une fois que l'instance de la classe est créée, il devient inutile
  - Diminue les performances de l'application
- Pour améliorer le multithreading
  - Ne rien faire si la performance de getInstance n'est pas vitale pour votre application
  - Préférer une instance créée au démarrage à une instance créée à la demande
    - Initialisation explicite de l'instance (initialisation simple ou avec des blocs statiques)

Le pattern Singleton et le multithreading

- Pour améliorer le multithreading
  - Il est également possible d'utiliser le verrouillage à double vérifications

```
public class Singleton
{
    private volatile static Singleton uniqueInstance;
    public static Singleton getInstance()
    {
        if(uniqueInstance == null)
        {
            synchronized(Singleton.class) {
                if (uniqueInstance == null)
                {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}
```