# Singleton Design pattern

- Singleton Design pattern is one of the simplest design patterns. It ensures a class only has one instance, and provides a global point of access to it.
- A design pattern that restricts the instantiation of class to one "single" object.
- Throughout the lifetime of the application the instance will remain same.
- Class's constructor should be private.
- Instance should be requested instead of created

# Why we need Singleton Design pattern?

- When there is single resource throughout the application, example database, log file etc.

# How to create a singleton class?

- 
- **Private Constructor:** The Singleton pattern incorporates a private constructor; this ensures that the class has control over its instantiation process.
- Static method: create object of class if it is not created. If created, then return previously created object.
- **Static Member:** The Singleton pattern employs a static member within the class. This static member ensures that memory is allocated only once, preserving the single instance of the Singleton class.
- Singleton class can be instantiated by two methods:
- **Early initialization:** In this method, class is initialized whether it is to be used or not. The main advantage of this method is its simplicity. You initiate the class at the time of class loading. Its drawback is that class is always initialized whether it is being used or not.
- **Lazy initialization:** In this method, class in initialized only when it is required. It can save you from instantiating the class when you don't need it. Generally, lazy initialization is used when we create a singleton class.

```
/*package whatever //do not write package name here */
import java.io.*;
class Singleton {
    // static class
    private static Singleton instance;
    private Singleton()
    {
        System.out.println("Singleton is Instantiated.");
    }
    public static Singleton getInstance()
    {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
    public static void doSomething()
    {
        System.out.println("Somethong is Done.");
    }
}

class GFG {
    public static void main(String[] args)
    {
        Singleton.getInstance().doSomething();
    }
}
```

# Why the constructor is private?

We don't want to allow the user to create the instance of class many times. So we make it private