# Singleton Class in Java

In Java or any other object-oriented programming language, singleton class is one that has only one object of a class (an instance) at a time. To make a class singleton:

1. Make the constructor private
2. Define a public static method that has the return type of object of the singleton class and initialises and returns the class instance variable
3. Declare a private static instance of the same class

```
public class Singleton {

  private static Singleton singleton = new Singleton( );

  /* A private Constructor prevents any other
   * class from instantiating.
   */
  private Singleton() { }

  /* Static 'instance' method */
  public static Singleton getInstance( ) {
    return singleton;
  }

  /* Other methods protected by singleton-ness */
  protected static void demoMethod( ) {
    System.out.println("demoMethod for singleton");
  }
public class SingletonDemo {

  public static void main(String[] args) {
    Singleton tmp = Singleton.getInstance( );
    tmp.demoMethod( );
  }
}
```

Output:

```
demoMethod for singleton
```

When we try to re-instantiate the singleton class, the new variable also points to the first instance created. So, any changes in either of them gets reflected in the other.

## Thread Safety in Singleton Class

Singleton class is thread unsafe as two objects can be created by cloning i.e. implementing the cloneable interface. The following code snippet shows how it can be achieved:

```
  Class Singleton_Clone implements Cloneable {
     private static Singleton_Clone instance = null;
     private Singleton_Clone() {
     }
     public static Singleton_Clone getInstance() {
        if(instance==null)
        {
           instance = new Singleton_Clone();
        }
        return instance;
     }
     // This method can create 2 objects of a Singleton class
     @Override
     protected Object clone() throws CloneNotSupportedException {
        return super.clone();
     }
  }
```

## Double Checked Locking of Singleton

This is a way to ensure thread safety of a singleton class and ensure that only one instance of a class is created throughout the lifecycle of the application. It is achieved by checking for the existence of a Singleton class instance twice with and without locking to double ensure that no more that a single instance of a Singleton class is created.

```
class SingletonThreadSafe{

    private static volatile SingletonThreadSafe
object=null;
    private static Object mutex=new Object();
    private SingletonThreadSafe(){
    }
    public static SingletonThreadSafe
getSingletonThreadSafe(){
        SingletonThreadSafe instance=object;
        if(instance==null)
        {
            synchronized (mutex)
            {
                instance=object;
                if(instance==null)
                    object=instance=new
SingletonThreadSafe();
            }
        }
        return instance;
    }
}
```