public class SingletonClass{

    private static volatile SingletonClass instance=null;

    public static SingletonClass getInstance(){

            synchronized(SingletonClass.class){

                if(instance==null){

                    instance=new SingletonClass();

                }

            }

        return instance;

    }

    private SingletonClass(){}

On my quest for the truth I discovered that there are **actually very few** "acceptable" reasons to use a Singleton.

One reason that tends to come up over and over again on the internets is that of a "logging" class (which you mentioned). In this case, a Singleton can be used instead of a single instance of a class because a logging class usually needs to be used **over and over again ad nauseam by every class** in a project. If every class uses this logging class, dependency injection becomes cumbersome.