Singleton Design Pattern

singleton pattern is a design pattern where only one instance of the class exists throught life time of the application program.

No matter how many times create object it will return the same instance.

Singleton design pattern can achieve by declaring constructor has private so as to restrict instance creation by the outside world.   Instead we expose a static getInstance API which would take care of object creation.

```
#include <iostream>
#include <mutex>
using namespace std;

 class Singleton
 {
     private:
            Singleton()
             {

              }
           static Singleton* instance;
           static std::mutex m_;
     public:
      static Singleton* getInstance();
      void testMethod();
      // destructor purpose
      static Singleton* destroyInstance();
 };

Singleton* Singleton::instance=nullptr;

 Singleton* Singleton::getInstance()
 {
     if(instance == nullptr) {
       mutex.lock();
        if(instance == nullptr){
          instance = new Singleton();
        }
        mutex.unlock();
      }
    return instance;
  }
   void Singleton::testMethod()
   {
          cout<<”test method in Singleton”<<endl;
        }
        Singleton* Singleton::destroyInstance()
        {
            delete instance;
            instance == nullptr;
         }


int main()
{
   Singleton* obj = Singleton::getInstance();
   obj→testMethod();

   obj-> destroyInstance();

   return 0;
}
```

If we delete in destructor it will cause infinite loop will happen. Destructor will call again destructor.

Limitations:

Global access to singleton object is bad design practice[because you hide the dependencies of your application in your code so instead of exposing them through interfaces]

Singleton design pattern doesn’t support inheritance of the class.