

Singleton

Creational Pattern

, private , .

Singleton
-static uniqueInstance -singletonData
+static instance() +SingletonOperation()

[singleton1.cpp](#)

```
class StringSingleton
{
public:
    // Some accessor functions for the class, itself
    std::string GetString() const
    {return mString;}
    void SetString(const std::string &newStr)
    {mString = newStr;}

    // The magic function, which allows access to the class from
    anywhere
```

```
// To get the value of the instance of the class, call:
//   StringSingleton::Instance().GetString();
static StringSingleton &Instance()
{
    // This line only runs once, thus creating the only instance in
existence
    static StringSingleton *instance = new StringSingleton;
    // dereferencing the variable here, saves the caller from
having to use
    // the arrow operator, and removes temptation to try and delete
the
    // returned instance.
    return *instance; // always returns the same instance
}

private:
    // We need to make some given functions private to finish the
definition of the singleton
    StringSingleton(){} // default constructor available only to
members or friends of this class

    // Note that the next two functions are not given bodies, thus any
attempt
    // to call them implicitly will return as compiler errors. This
prevents
    // accidental copying of the only instance of the class.
    StringSingleton(const StringSingleton &old); // disallow copy
constructor
    const StringSingleton &operator=(const StringSingleton &old);
//disallow assignment operator

    // Note that although this should be allowed,
    // some compilers may not implement private destructors
    // This prevents others from deleting our one single instance,
which was otherwise created on the heap
    ~StringSingleton(){}
private: // private data for an instance of this class
    std::string mString;
};
```

singleton2.cpp

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

/* Place holder for thread synchronization mutex */
class Mutex
{
    /* placeholder for code to create, use, and free a mutex */
};
```

```
/* Place holder for thread synchronization lock */
class Lock
{ public:
    Lock(Mutex& m) : mutex(m) { /* placeholder code to acquire the
mutex */ }
    ~Lock() { /* placeholder code to release the mutex */ }
private:
    Mutex & mutex;
};

class Singleton
{ public:
    static Singleton* GetInstance();
    int a;
    ~Singleton() { cout << "In Destructor" << endl; }

private:
    Singleton(int _a) : a(_a) { cout << "In Constructor" << endl;
}

    static Mutex mutex;

    // Not defined, to prevent copying
    Singleton(const Singleton& );
    Singleton& operator =(const Singleton& other);
};

Mutex Singleton::mutex;

Singleton* Singleton::GetInstance()
{
    Lock lock(mutex);

    cout << "Get Instance" << endl;

    // Initialized during first access
    static Singleton inst(1);

    return &inst;
}

int main()
{
    Singleton* singleton = Singleton::GetInstance();
    cout << "The value of the singleton: " << singleton->a << endl;
    return 0;
}
```

http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Singleton

From:

<http://obg.co.kr/doku/> - **OBG WiKi**

Permanent link:

http://obg.co.kr/doku/doku.php?id=programming:design_pattern:singleton

Last update: **2020/11/29 14:09**

