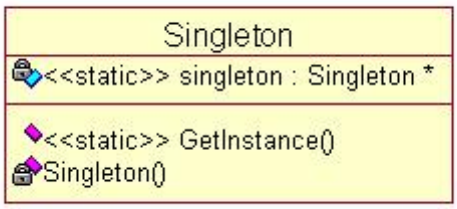
软件领域中的设计模式为开发人员提供了一种使用专家设计经验的有效途径。设计模式中运用了面向对象编程语言的重要特性：封装、继承、多态，真正领悟设计模式的精髓是可能一个漫长的过程，需要大量实践经验的积累。最近看设计模式的书，对于每个模式，用 C++写了个小例子，加深一下理解。主要参考《大话设计模式》和《设计模式:可复用面向对象软件的基础》（DP）两本书。本文介绍单例模式的实现。

单例的一般实现比较简单，下面是代码和 UML 图。由于构造函数是私有的，因此无法通过构造函数实例化，唯一的方法就是通过调用静态函数 GetInstance。

UML 图：



代码：

[cpp] view plain copy print?

```cpp
//Singleton.h
class Singleton
{
public:
    static Singleton* GetInstance();
private:
    Singleton() {}
    static Singleton *singleton;
};
//Singleton.cpp
Singleton* Singleton::singleton = NULL;
Singleton* Singleton::GetInstance()
{
    if(singleton == NULL)
        singleton = new Singleton();
    return singleton;
}
```

这里只有一个类，如何实现 Singleton 类的子类呢？也就说 Singleton 有很多子类，在一种应用中，只选择其中的一个。最容易就是在 GetInstance 函数中做判断，比如可以传递一个字符串，根据字符串的内容创建相应的子类实例。这也是 DP 书上的一种解法，书上给的代码不全。这里重新实现了一下，发现不是想象中的那么简单，最后实现的版本看上去很怪异。在 VS2008 下测试通过。

[cpp] view plain copy print?

```cpp
//Singleton.h
#pragma once
#include <iostream>
using namespace std;

class Singleton
{
public:
    static Singleton* GetInstance(const char* name);
    virtual void Show() {}
protected: //必须为保护，如果是私有属性，子类无法访问父类的构造函数
    Singleton() {}
private:
    static Singleton *singleton; //唯一实例的指针
};

//Singleton.cpp
#include "Singleton.h"
#include "SingletonA.h"
#include "SingletonB.h"
Singleton* Singleton::singleton = NULL;
Singleton* Singleton::GetInstance(const char* name)
{
    if(singleton == NULL)
    {
        if(strcmp(name, "SingletonA") == 0)
            singleton = new SingletonA();
        else if(strcmp(name,"SingletonB") == 0)
            singleton = new SingletonB();
        else
            singleton = new Singleton();
    }
    return singleton;
}
```

[cpp] view plain copy print?

```cpp
//SingletonA.h
#pragma once
#include "Singleton.h"
class SingletonA: public Singleton
{
    friend class Singleton; //必须为友元类，否则父类无法访问子类的构造函数
public:
    void Show() { cout<<"SingletonA"<<endl; }
```

```
9.  private:   //为保护属性，这样外界无法通过构造函数进行实例化
10.     SingletonA() {}
11. };
12. //SingletonB.h
13. #pragma once
14. #include "Singleton.h"
15. class SingletonB: public Singleton
16. {
17.     friend class Singleton; //必须为友元类，否则父类无法访问子类的构造函数
18. public:
19.     void Show(){ cout<<"SingletonB"<<endl; }
20. private:  //为保护属性，这样外界无法通过构造函数进行实例化
21.     SingletonB() {}
22. };
```

[cpp] view plain copy print?

```
1.  #include "Singleton.h"
2.  int main()
3.  {
4.      Singleton *st = Singleton::GetInstance("SingletonA");
5.      st->Show();
6.      return 0;
7.  }
```

上面代码有一个地方很诡异，父类为子类的友元，如果不是友元，函数 GetInstance 会报错，意思就是无法调用 SingletonA 和 SIngletonB 的构造函数。父类中调用子类的构造函数，我还是第一次碰到。当然了把 SingletonA 和 SIngletonB 的属性设为 public，GetInstance 函数就不会报错了，但是这样外界就可以定义这些类的对象，违反了单例模式。

看似奇怪，其实也容易解释。在父类中构建子类的对象，相当于是外界调用子类的构造函数，因此当子类构造函数的属性为私有或保护时，父类无法访问。为共有时，外界就可以访问子类的构造函数了，此时父类当然也能访问了。只不过为了保证单例模式，所以子类的构造函数不能为共有，但是又希望在父类中构造子类的对象，即需要调用子类的构造函数，这里没有办法才出此下策：将父类声明为子类的友元类。