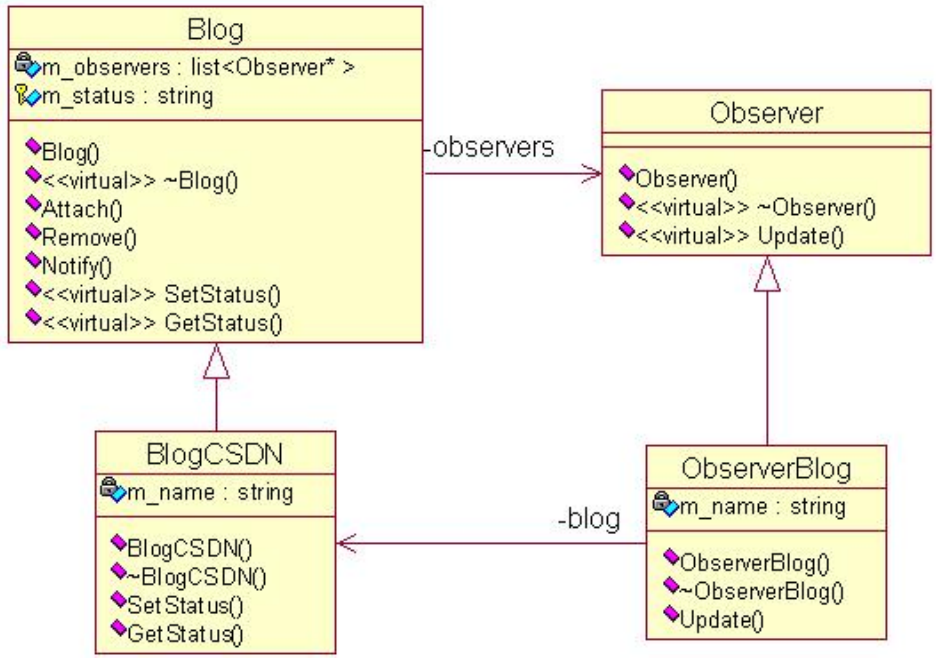
软件领域中的设计模式为开发人员提供了一种使用专家设计经验的有效途径。设计模式中运用了面向对象编程语言的重要特性：封装、继承、多态，真正领悟设计模式的精髓是可能一个漫长的过程，需要大量实践经验的积累。最近看设计模式的书，对于每个模式，用 C++写了个小例子，加深一下理解。主要参考《大话设计模式》和《设计模式:可复用面向对象软件的基础》两本书。本文介绍观察者模式的实现。

观察者模式：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。它还有两个别名，依赖(Dependents)，发布-订阅(Publish-Subsrcibe)。可以举个博客订阅的例子，当博主发表新文章的时候，即博主状态发生了改变，那些订阅的读者就会收到通知，然后进行相应的动作，比如去看文章，或者收藏起来。博主与读者之间存在种一对多的依赖关系。下面给出相应的 UML 图设计。



可以看到博客类中有一个观察者链表（即订阅者），当博客的状态发生变化时，通过 Notify 成员函数通知所有的观察者，告诉他们博客的状态更新了。而观察者通过 Update 成员函数获取博客的状态信息。代码实现不难，下面给出 C++的一种实现。

[cpp] view plain copy print?

```cpp
//观察者
class Observer
{
public:
    Observer() {}
    virtual ~Observer() {}
```

```
7.      virtual void Update() {}
8.  };
9.  //博客
10. class Blog
11. {
12. public:
13.     Blog() {}
14.     virtual ~Blog() {}
15.     void Attach(Observer *observer) { m_observers.push_back(observer); }
    //添加观察者
16.     void Remove(Observer *observer) { m_observers.remove(observer); }
    //移除观察者
17.     void Notify() //通知观察者
18.     {
19.         list<Observer*>::iterator iter = m_observers.begin();
20.         for(; iter != m_observers.end(); iter++)
21.             (*iter)->Update();
22.     }
23.     virtual void SetStatus(string s) { m_status = s; } //设置状态
24.     virtual string GetStatus() { return m_status; }    //获得状态
25. private:
26.     list<Observer* > m_observers; //观察者链表
27. protected:
28.     string m_status; //状态
29. };
```

以上是观察者和博客的基类，定义了通用接口。博客类主要完成观察者的添加、移除、通知操作，设置和获得状态仅仅是一个默认实现。下面给出它们相应的子类实现。

[cpp] view plain copy print?

```
1.  //具体博客类
2.  class BlogCSDN : public Blog
3.  {
4.  private:
5.      string m_name; //博主名称
6.  public:
7.      BlogCSDN(string name): m_name(name) {}
8.      ~BlogCSDN() {}
9.      void SetStatus(string s) { m_status = "CSDN 通知 : " + m_name + s; } //具
    体设置状态信息
10.     string GetStatus() { return m_status; }
11. };
12. //具体观察者
13. class ObserverBlog : public Observer
```

```
{
private:
    string m_name;  //观察者名称
    Blog *m_blog;   //观察的博客，当然以链表形式更好，就可以观察多个博客
public:
    ObserverBlog(string name,Blog *blog): m_name(name), m_blog(blog) {}
    ~ObserverBlog() {}
    void Update()  //获得更新状态
    {
        string status = m_blog->GetStatus();
        cout<<m_name<<"-------"<<status<<endl;
    }
};
```

客户的使用方式：

[cpp] view plain copy print?

```
//测试案例
int main()
{
    Blog *blog = new BlogCSDN("wuzhekai1985");
    Observer *observer1 = new ObserverBlog("tutupig", blog);
    blog->Attach(observer1);
    blog->SetStatus("发表设计模式 C++实现（15）——观察者模式");
    blog->Notify();
    delete blog; delete observer1;
    return 0;
}
```