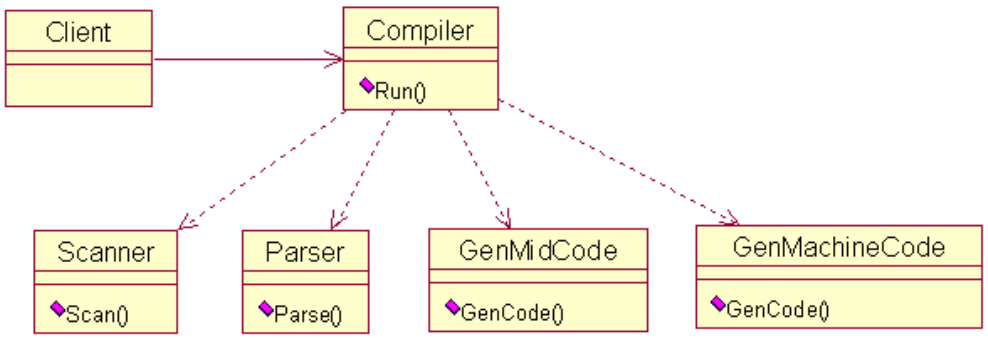
設計模式C++實現（7）——外觀模式、組合模式

星期六, 2013 12月 14, 1:04 上午

軟件領域中的設計模式為開發人員提供了一種使用專家設計經驗的有效途徑。設計模式中運用了面向對象編程語言的重要特性：封裝、繼承、多態，真正領悟設計模式的精髓是可能一個漫長的過程，需要大量實踐經驗的積累。最近看設計模式的書，對於每個模式，用C++寫了個小例子，加深一下理解。主要參考《大話設計模式》和《設計模式:可復用面向對象軟件的基礎》（DP）兩本書。本文介紹外觀模式和組合模式的實現。

外觀模式應該是用的很多的一種模式，特別是當一個系統很複雜時，系統提供給客戶的是一個簡單的對外接口，而把裡面複雜的結構都封裝了起來。客戶只需使用這些簡單接口就能使用這個系統，而不需要關注內部複雜的結構。DP一書的定義：為子系統中的一組接口提供一個一致的界面， 外觀模式定義了一個高層接口，這個接口使得這一子系統更加容易使用。舉個編譯器的例子，假設編譯一個程序需要經過四個步驟：詞法分析、語法分析、中間代碼生成、機器碼生成。學過編譯都知道，每一步都很複雜。對於編譯器這個系統，就可以使用外觀模式。可以定義一個高層接口，比如名為Compiler的類，裡面有一個名為Run的函數。客戶只需調用這個函數就可以編譯程序，至於Run函數內部的具體操作，客戶無需知道。下面給出UML圖，以編譯器為實例。



相應的代碼實現為：

```
class Scanner
{
public:
    void Scan() { cout<<"詞法分析"<<endl; }
};
class Parser
{
public:
    void Parse() { cout<<"語法分析"<<endl; }
};
class GenMidCode
{
public:
    void GenCode() { cout<<"產生中間代碼"<<endl; }
};
class GenMachineCode
{
public:
    void GenCode() { cout<<"產生機器碼"<<endl;}
};
//高層接口
class Compiler
{
public:
    void Run()
    {
        Scanner scanner;
        Parser parser;
        GenMidCode genMidCode;
        GenMachineCode genMacCode;
        scanner.Scan();
        parser.Parse();
        genMidCode.GenCode();
        genMacCode.GenCode();
    }
};
```

客戶使用方式：

```
1. int main()
2. {
3.     Compiler compiler;
4.     compiler.Run();
5.     return 0;
6. }
```

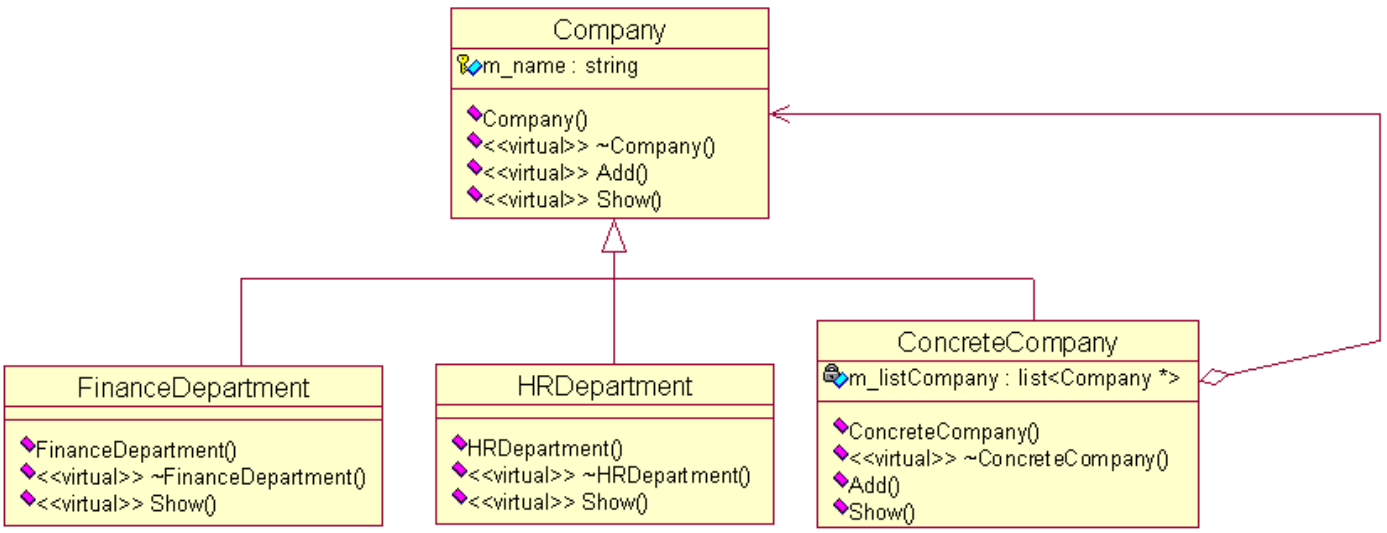
這就是外觀模式，它有幾個特點（摘自DP一書），（1）它對客戶屏蔽子系統組件，因而減少了客戶處理的對象的數目並使得子系統使用起來更加方便。（2）它實現了子系統與客戶之間的松耦合關係，而子系統內部的功能組件往往是緊耦合的。（3）如果應用需要，它並不限制它們使用子系統類。

結合上面編譯器這個例子，進一步說明。對於（1），編譯器類對客戶屏蔽了子系統組件，客戶只需處理編譯器的對象就可以方便的使用子系統。對於（2），子系統的變化，不會影響到客戶的使用，體現了子系統與客戶的松耦合關係。對於（3），如果客戶希望使用詞法分析器，只需定義詞法分析的類對象即可，並不受到限制。

外觀模式在構建大型系統時非常有用。接下來介紹另一種模式，稱為組合模式。感覺有點像外觀模式，剛才我們實現外觀模式時，在Compiler這個類中包含了多個類的對象，就像把這些類組合在了一起。組合模式是不是這個意思，有點相似，其實不然。

DP書上給出的定義：將對象組合成樹形結構以表示「部分-整體」的層次結構。組合使得用戶對單個對象和組合對象的使用具有一致性。注意兩個字「樹形」。這種樹形結構在現實生活中隨處可見，比如一個集團公司，它有一個母公司，下設很多家子公司。不管是母公司還是子公司，都有各自直屬的財務部、人力資源部、銷售部等。對於母公司來說，不論是子公司，還是直屬的財務部、人力資源部，都是它的部門。整個公司的部門拓撲圖就是一個樹形結構。

下面給出組合模式的UML圖。從圖中可以看到，FinanceDepartment、HRDepartment兩個類作為葉結點，因此沒有定義添加函數。而ConcreteCompany類可以作為中間結點，所以可以有添加函數。那麼怎麼添加呢？這個類中定義了一個鏈表，用來放添加的元素。



相應的代碼實現為：

```
1. class Company
2. {
3. public:
4.     Company(string name) { m_name = name; }
5.     virtual ~Company(){}
6.     virtual void Add(Company *pCom){}
7.     virtual void Show(int depth) {}
8. protected:
9.     string m_name;
10. };
11. //具體公司
12. class ConcreteCompany : public Company
13. {
14. public:
15.     ConcreteCompany(string name): Company(name) {}
16.     virtual ~ConcreteCompany() {}
```

```
    void Add(Company *pCom) { m_listCompany.push_back(pCom); } //位於樹的中間，可以增加子樹
    void Show(int depth)
    {
        for(int i = 0;i < depth; i++)
            cout<<"-";
        cout<<m_name<<endl;
        list<Company *>::iterator iter=m_listCompany.begin();
        for(; iter != m_listCompany.end(); iter++) //顯示下層結點
            (*iter)->Show(depth + 2);
    }
private:
    list<Company *> m_listCompany;
};
//具體的部門，財務部
class FinanceDepartment : public Company
{
public:
    FinanceDepartment(string name):Company(name){}
    virtual ~FinanceDepartment() {}
    virtual void Show(int depth) //只需顯示，無限添加函數，因為已是葉結點
    {
        for(int i = 0; i < depth; i++)
            cout<<"-";
        cout<<m_name<<endl;
    }
};
//具體的部門，人力資源部
class HRDepartment :public Company
{
public:
    HRDepartment(string name):Company(name){}
    virtual ~HRDepartment() {}
    virtual void Show(int depth) //只需顯示，無限添加函數，因為已是葉結點
    {
        for(int i = 0; i < depth; i++)
            cout<<"-";
        cout<<m_name<<endl;
    }
};
```

客戶使用方式：

```
int main()
{
    Company *root = new ConcreteCompany("總公司");
    Company *leaf1=new FinanceDepartment("財務部");
    Company *leaf2=new HRDepartment("人力資源部");
    root->Add(leaf1);
    root->Add(leaf2);

    //分公司A
    Company *mid1 = new ConcreteCompany("分公司A");
    Company *leaf3=new FinanceDepartment("財務部");
    Company *leaf4=new HRDepartment("人力資源部");
    mid1->Add(leaf3);
    mid1->Add(leaf4);
    root->Add(mid1);
    //分公司B
    Company *mid2=new ConcreteCompany("分公司B");
    FinanceDepartment *leaf5=new FinanceDepartment("財務部");
    HRDepartment *leaf6=new HRDepartment("人力資源部");
    mid2->Add(leaf5);
    mid2->Add(leaf6);
    root->Add(mid2);
    root->Show(0);

    delete leaf1; delete leaf2;
    delete leaf3; delete leaf4;
    delete leaf5; delete leaf6;
    delete mid1; delete mid2;
    delete root;
```

```
30.     return 0;
31. }
```

上面的實現方式有缺點，就是內存的釋放不好，需要客戶自己動手，非常不方便。有待改進，比較好的做法是讓ConcreteCompany類來釋放。因為所有的指針都是存在ConcreteCompany類的鏈表中。C++的麻煩，沒有垃圾