AlexOne

五、装饰模式

目录

1 传统方法的实现

```
#include <iostream>
// 多态
using namespace std;
// 通过GrandFather的指针Son的对象，实现GrandFather的效果，该如何做到？？？
class GradFather{
public:
    virtual void test(){
        cout<<"GradFather test"<<endl;
    }
};

class Father:public GradFather{
public:
    virtual void test(){
        // GradFather::test();              // 两个地方都写上，会调用两次
        cout<<"Father test"<<endl;
    }
};
class Son:public Father  {
public:
    void test(){                            // 不能直接写test();
        GradFather::test();                 // 再次调用父类的虚函数
        Father::test();                     // 调用父类的虚函数，执行子类的实现
        cout<<"Son test"<<endl;
    }
};

int main(){
    // Father *pf=new Son;
    // pf->test();

    GradFather *pg = new Son;
    pg->test();                             // 调用父类的虚函数，执行子类的实现

    return 0;
}
```

2 实例

```cpp
#include <iostream>
using namespace std;
/*
 抽象基类Phone出来分了两条路，一条由Nokia继承泛化，另一条由各种装饰品继承泛化
 本实例模拟卖手机，手机和保护套、耳机、贴膜等有不同的搭配，
 使用装饰模式去计算价格。动态的给一个对象添加一些额外的职责。
 注意，是 [ 动态的为一个对象添加新功能而不是为类添加新功能 ]
*/
class Phone{                                    // 首先抽象出一个手机类
public:
    virtual int cost()=0;
};

class Nokia:public Phone{
public:
    int cost(){
        return 5000;
    }
};

class DecoratePhone:public Phone{           // 构造一个装饰器类，接收传入的手机类指针
public:                                     // 将手机嵌入到另一对象中
    DecoratePhone(Phone *ph):phone(ph){}    // 默认构造+初始化列表
protected:
    Phone *phone;                           // 装饰器将被装饰者(手机)对象指针私有化
};

class ScreenProtectorPhone:public DecoratePhone{
public:
    ScreenProtectorPhone(Phone * ph):DecoratePhone(ph){}
    int cost(){
        return 100 + phone->cost();         // 注意这个phone是在父类里面定义的
    }
};

class HeadSetPhone : public DecoratePhone{
public:
    HeadSetPhone(Phone *ph): DecoratePhone(ph){}
    int cost(){
        return 200 + phone->cost();
    }
};

int main(){
    Nokia nk;
    cout<< nk.cost()<<endl;

    ScreenProtectorPhone sp(&nk);
    cout<<sp.cost()<<endl;
    //一件商品买一次，再买无效
    Phone * p=
        new HeadSetPhone(
            new ScreenProtectorPhone(
                new HeadSetPhone(
                    new ScreenProtectorPhone(
                        new Nokia))));      // 如参是phone，实际上是phone的各种子类函数
                        // 确实是多态的应用。Phone为抽象基类，各种装饰品是DecoratePhone的继承
                        // Nokia是对phone的扩展
    cout<<p->cost()<<endl;
}
```