Decorator 패턴

타입

Structural Pattern

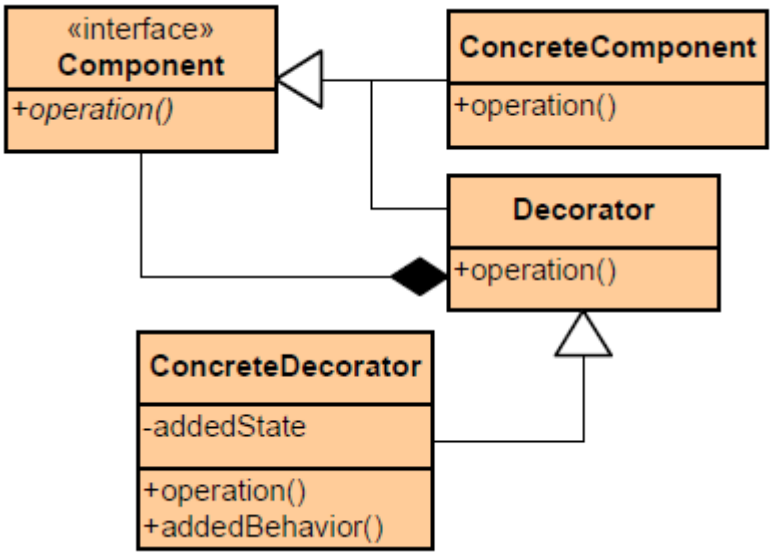
문제

이미 정의되어 있는 클래스에 특정 메소드를 동적으로 추가하고 싶다.

해결

기본 클래스를 상속받아 decorator 클래스를 만들고 새로 추가될 기능은 decorator 클래스를 상속받아 클래스에 정의한다.

클래스 다이어그램



예제

decorator.cpp

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

class Car //Our Abstract base class
```

```
{
protected:
    string _str;
public:
    Car()
    {
        _str = "Unknown Car";
    }

    virtual string getDescription()
    {
        return _str;
    }

    virtual double getCost() = 0;

    virtual ~Car()
    {
        cout << "~Car()\n";
    }
};

class OptionsDecorator : public Car //Decorator Base class
{
public:
    virtual string getDescription() =0;

    virtual ~OptionsDecorator()
    {
        cout<<"~OptionsDecorator()\n";
    }
};


class CarModel1 : public Car
{
public:
    CarModel1()
    {
        _str = "CarModel1";
    }
    virtual double getCost()
    {
        return 31000.23;
    }

    ~CarModel1()
    {
        cout<<"~CarModel1()\n";
    }
```

```
};


class Navigation: public OptionsDecorator
{
    Car *_b;
public:
    Navigation(Car *b)
    {
        _b = b;
    }
    string getDescription()
    {
        return _b->getDescription() + ", Navigation";
    }

    double getCost()
    {
        return 300.56 + _b->getCost();
    }
    ~Navigation()
    {
        cout << "~Navigation()\n";
    }
};

class PremiumSoundSystem: public OptionsDecorator
{
    Car *_b;
public:
    PremiumSoundSystem(Car *b)
    {
        _b = b;
    }
    string getDescription()
    {
        return _b->getDescription() + ", PremiumSoundSystem";
    }

    double getCost()
    {
        return 0.30 + _b->getCost();
    }
    ~PremiumSoundSystem()
    {
        cout << "~PremiumSoundSystem()\n";
    }
};

class ManualTransmition: public OptionsDecorator
{
```

```
    Car *_b;
public:
    ManualTransmition(Car *b)
    {
        _b = b;
    }
    string getDescription()
    {
        return _b->getDescription()+ ", Soy Milk";
    }

    double getCost()
    {
        return 0.30 + _b->getCost();
    }
    ~ManualTransmition()
    {
        cout << "~ManualTransmition()\n";
    }
};

class CarDecoratorExample{
public:

    void execute()
    {
        //Create our Car that we want to buy
        Car *b = new CarModel1();

        cout << "Base model of " << b->getDescription() << " costs $"
<< b->getCost() << "\n";

        //Who wants base model lets add some more features

        b = new Navigation(b);
        cout << b->getDescription() << " will cost you $" <<
b->getCost() << "\n";
        b = new PremiumSoundSystem(b);
        b = new ManualTransmition(b);
        cout << b->getDescription() << " will cost you $" <<
b->getCost() << "\n";


        delete b;
    }

};


int main()
```

```
{
    CarDecoratorExample b;
    b.execute();
    return 0;
}
```

참고

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

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

Permanent link:
http://obg.co.kr/doku/doku.php?id=programming:design_pattern:decorator

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