Наблюдатель

Наблюдатель- определяет отношение “один ко-многтм” между объетаи таким образом, что при изменении состояния одного объекта происходит автоматическое оповещение и обновление всех зависимых объктов

- Субъекты обновляют наблюдателей через единый интерфейс
- Субъект ничего не знает о наблюдателях кроме того что они реализуют интерфейс Observer.
- При использовании пвттерна возможен как запрос так и активная доставка данных от субъекта
- Работа кода не должна зависеть от порядка оповещения

Используйте паттерн наблюдатель в следующих ситуациях:

Когда у абстракции есть два аспекта, один из которых зависит от другого. Инкапсуляции этих аспектов в разные объекты позволяют изменять и повторно использовать их независимо;

Когда при модификации одного объекта требуется изменить другие и вы не знаете, сколько именно объектов нужно изменить;

Когда один объект должен оповещать других, не делая предположений об уведомляемых объектах. Другими словами, вы не хотите, чтобы объекты были тесно связаны между собой.

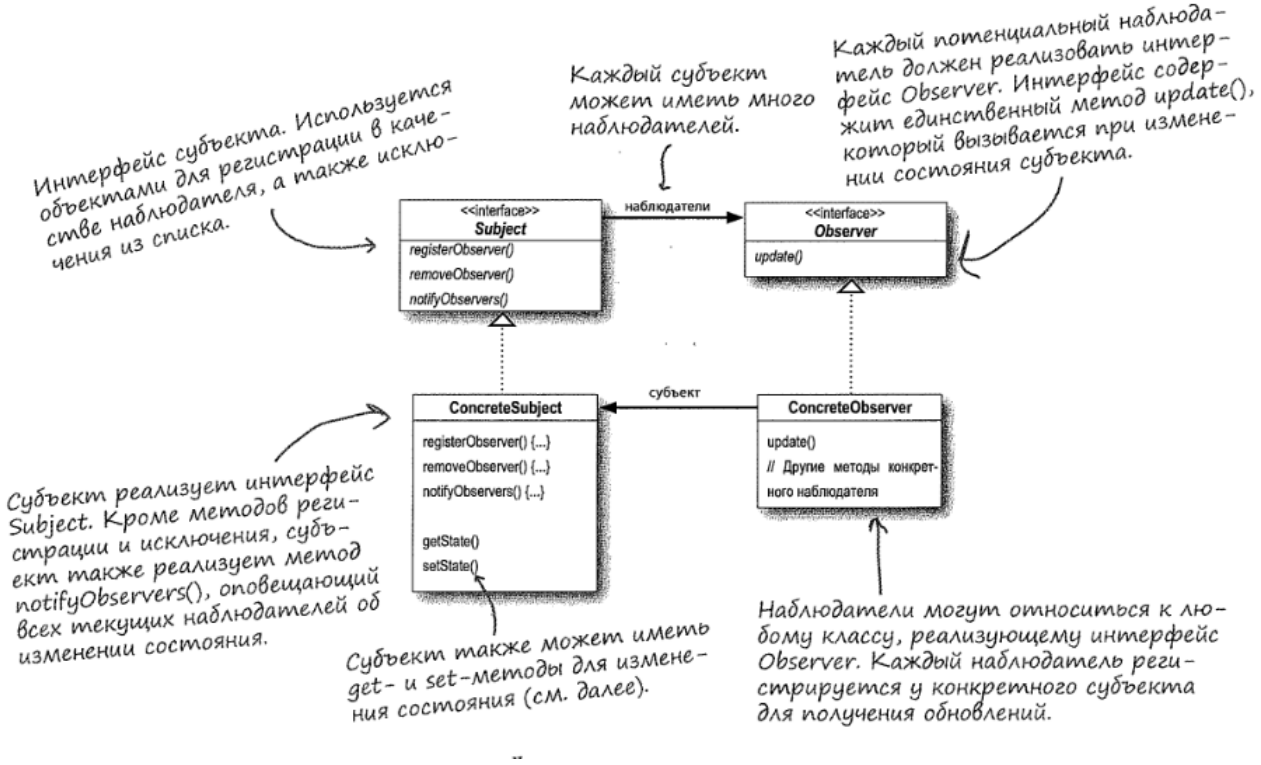
Результаты

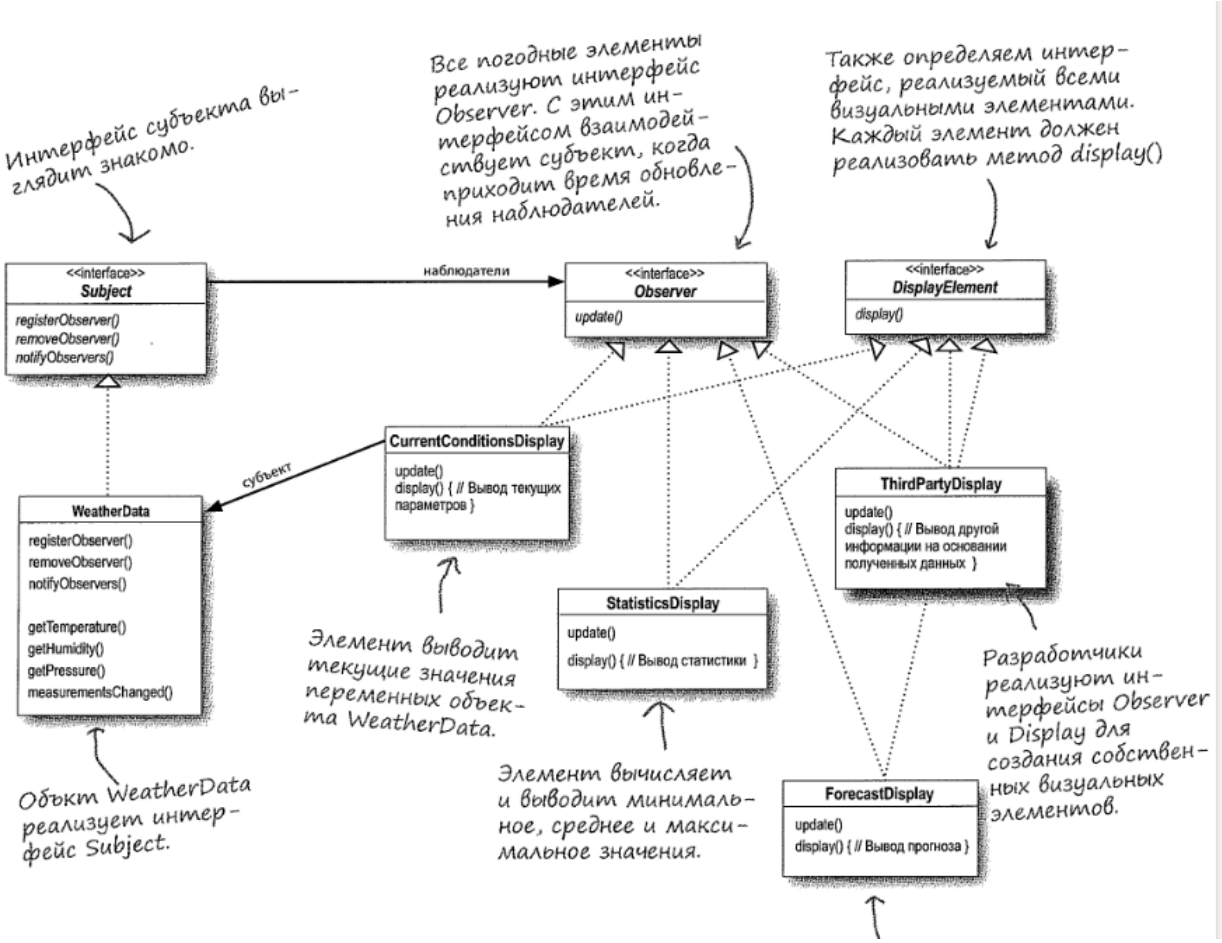
- Паттерн наблюдатель позволяет изменять субъекты и наблюдатели независимо друг от друга. Субъекты разрешается повторно использовать без участия наблюдателей, и наоборот. Это дает возможность добавлять
- новых наблюдателей без модификации субъекта или других наблюдателей.

Рассмотрим некоторые достоинства и недостатки паттерна наблюдатель:

- Абстрактная связанность субъекта и наблюдателя. Субъект имеет информацию лишь о том, что у него есть ряд наблюдателей, каждый из которых подчиняется простому интерфейсу абстрактного класса Observer.
- Поддержка широковещательных коммуникаций. В отличие от обычного запроса для уведомления, посылаемого субъектом, не нужно задавать определенного получателя. Уведомление автоматически поступает всем подписавшимся на него объектам.
- Неожиданные обновления. Поскольку наблюдатели не располагают информацией друг о друге, им неизвестно и о том, во что обходится изменение субъекта.

Паттерн Наблюдатель: диаграмма классов






```cpp
#include <iostream>
#include <list>
using namespace std;
struct Data
{
    float temp;
    float humidity;
    float pressure;
};
//наблюдатели или подпискичи Абстрактный класс
class Observer
{
public:
    virtual void update(float temp,float humidity,float pressure)=0;
};
//Субъект Абстрактный класс
class Subject
{
public:
    //регистрация
    virtual void registerObserver( Observer& O) = 0;
    //отписывание
    virtual void removeObserver( Observer& O) = 0;
    //оповещение наблюдателей об измение состояния субъекта
```

```
    virtual void notifyObservers() = 0;
};
class DisplayElements
{
public:
    virtual  void display()=0;
};
class WeatherData:public Subject
{
private:
    list<Observer*> _observers;
    float temp;
    float humidity;
    float pressure;
private:
    void notifyObservers()
    {
        for(list<Observer*>::iterator iter = _observers.begin(); iter !
= _observers.end(); ++iter)
        {
            (*iter)->update(temp,humidity,pressure);
        }
        for(auto iter : _observers)
        {
        }
    }
public:
    void registerObserver(Observer& ref)
    {
        _observers.push_back(&ref);
    }
    void removeObserver(Observer& ref)
    {
        _observers.remove(&ref);
    }
    void setData(float temp,float humidity,float pressure)
    {
        this->temp=temp;
        this->humidity=humidity;
        this->pressure=pressure;
        notifyObservers();
    }
};
class  CurrentConditionals:Observer,DisplayElements
{
private:
    float temp;
    float humidity;
    Subject* WD;
public:
    CurrentConditionals( Subject& wd)
    {
        this->WD=&wd;
        WD->registerObserver(*this);
```

```
    }
    void update(float temp,float humidity,float pressure)
    {
        this->temp=temp;
        this->humidity=humidity;
        display();
    }
    void display()
    {
        cout<< "class1 "<< this->temp<<" "<< this->humidity<<endl;
    }
};
class  LastConditionals:Observer,DisplayElements
{
private:
    float temp;
    float humidity;
    Subject* WD;
public:
    LastConditionals( Subject& wd)
    {
        this->WD=&wd;
        WD->registerObserver(*this);
    }
    void update(float temp,float humidity,float pressure)
    {
        this->temp=temp;
        this->humidity=humidity;
        display();
    }
    void display()
    {
        cout<< "class2 "<< this->temp<<" "<< this->humidity<<endl;
    }
};
int main() {
    WeatherData* wd=new WeatherData();
    CurrentConditionals* cur=new CurrentConditionals(*wd);
    LastConditionals* last=new LastConditionals(*wd);
    wd->setData(10,12,13);
    wd->setData(14,16,13);
    wd->setData(15,17,13);
    return 0;
}
```