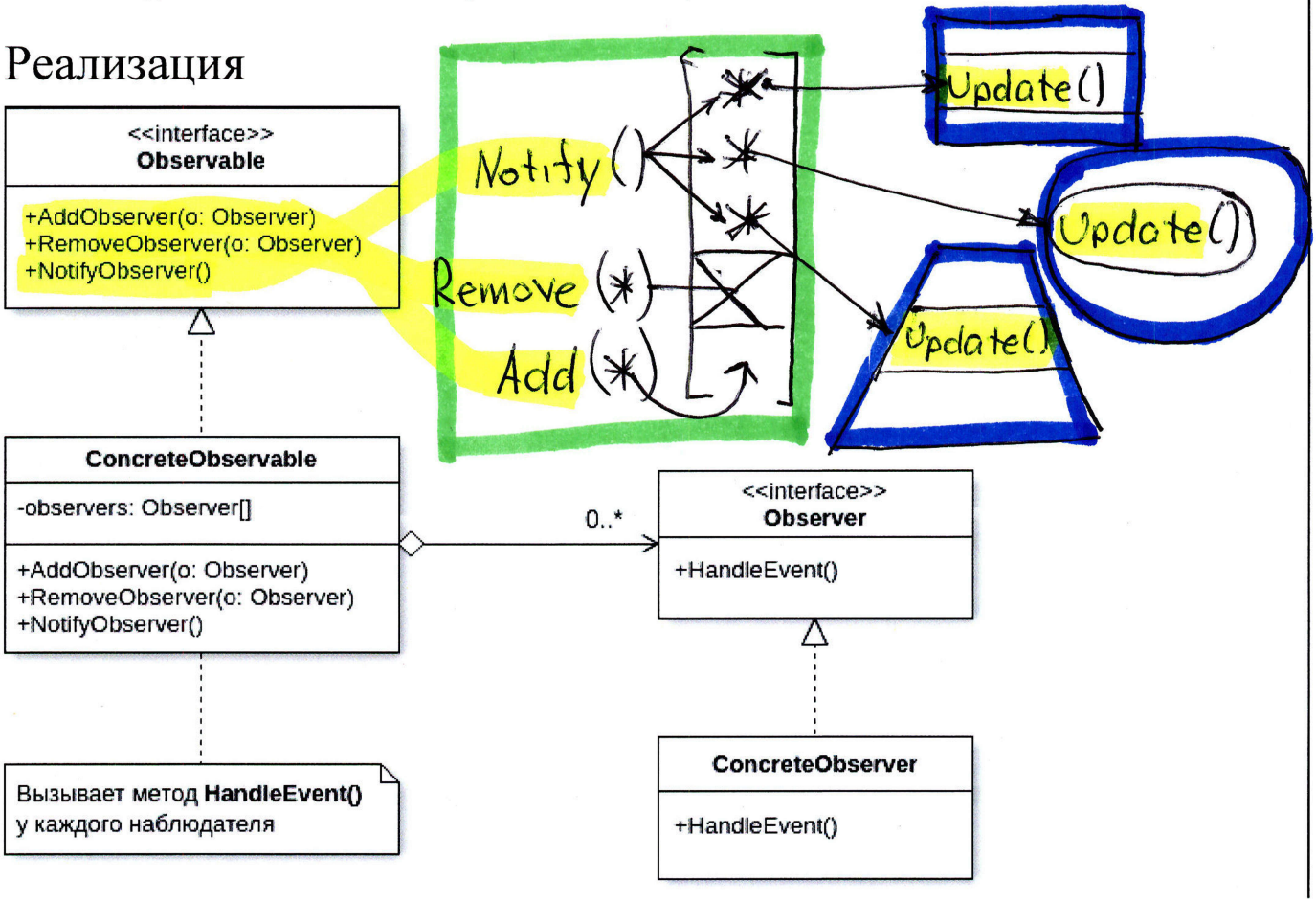
Наблюдатель (шаблон проектирования)

Наблюдатель (англ. *Observer*) — поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents). Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними. шаблон проектирования. Также известен как «подчинённые» (Dependents). Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.

Назначение

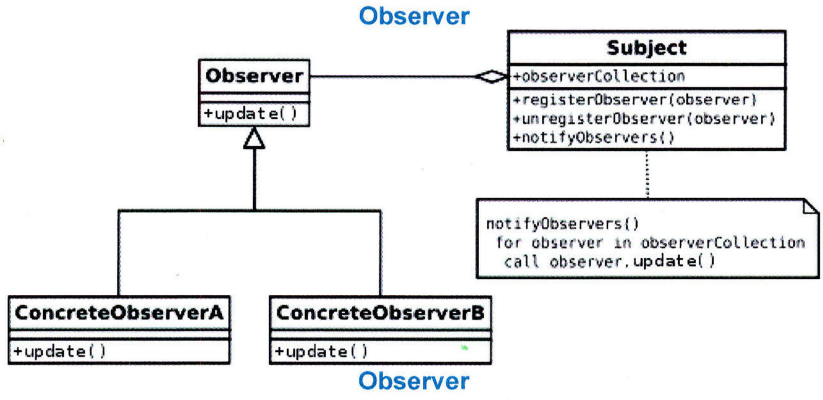
Определяет зависимость типа один ко многим между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

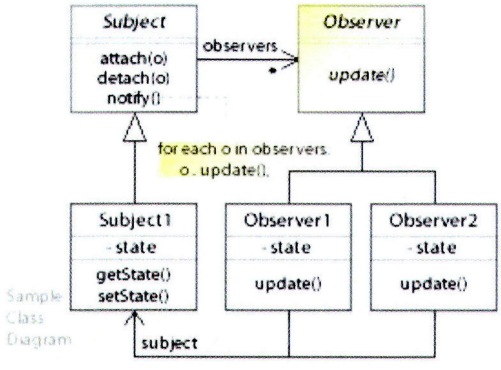
Реализация

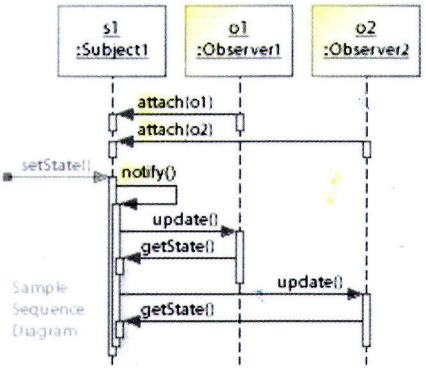


При реализации шаблона «наблюдатель» обычно используются следующие классы:

- ***IObservable*** — интерфейс, определяющий методы для добавления, удаления и оповещения наблюдателей;
- **ConcreteObservable** — конкретный класс, который реализует интерфейс ***IObservable***;
- ***IObserver*** — интерфейс, с помощью которого наблюдатель получает оповещение;
- **ConcreteObserver** — конкретный класс, который реализует интерфейс ***IObserver***.







Область применения

Шаблон «наблюдатель» применяется в тех случаях, когда система обладает следующими свойствами:

- существует как минимум один объект, рассылающий сообщения;
- имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения;
- позволяет избежать сильного зацепления) взаимодействующих классов.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.
http://www.cyberforum.ru/csharp-beginners/thread1831190.html

```
using System;
using System.Collections.Generic;

namespace ObserverSample
{

class Program
{
static void Main(string[] args)
{var subjectMagazine = new SubjectMagazine(); // создаём издателя

var observer1 = new Observer("001"); // создаём первого подписчика "001" и
subjectMagazine.RegisterObserver(observer1);  // подписываем его к Журналу

// Журнал оповещает всех имеющихся подписчиков
subjectMagazine.NotifyObservers(DateTime.Now);
Console.WriteLine();

var observer2 = new Observer("2"); // создаём второго подписчика и
subjectMagazine.RegisterObserver(observer2); // подписываем его к Журналу

// Ждём следущую новость в течении 2.5 секунд
System.Threading.Thread.Sleep(2500);
// и затем Журнал оповещает всех подписчиков
subjectMagazine.NotifyObservers(DateTime.Now);
}}

//////////////////////////////////////////////////
public interface IObservable
{
void RegisterObserver(IObserver observer);
void RemoveObserver(IObserver observer);
void NotifyObservers(DateTime dateTime);
}
//////////////////////////////////////
// Издатель с interface IObservable
public class SubjectMagazine : IObservable
{
private readonly IList<IObserver> _observers = new List<IObserver>(); * * * ...

public void RegisterObserver(IObserver observer)
{_observers.Add(observer); }
public void RemoveObserver(IObserver observer)
{int i = _observers.IndexOf(observer);
if(i >= 0) _observers.Remove(observer); }
public void NotifyObservers(DateTime dateTime)
{foreach(var observer in _observers)
        observer.Update(dateTime);
}}

//////////////////////////////////////////////////
public interface IObserver
{
void Update(DateTime dateTime);
}
//////////////////////////////////////
// подписчик с interface IObserver
public class Observer : IObserver
{
public string Name;

public Observer(string name)
{Name = name;}

public void Update(DateTime dateTime)
{Console.WriteLine(Name + " : " + dateTime);}
}
}

//////////////////////////////////////////////////
// вывод от первого подписчика
001 : 6/30/2019 4:56:31 PM

// спустя 2.5 секудны вывод от обоих подписчиков
001 : 6/30/2019 4:56:33 PM
2 : 6/30/2019 4:56:33 PM
```

```python
from abc import ABCMeta, abstractmethod

##########################################
class Observable(metaclass=ABCMeta):
    """Interface Абстрактный наблюдаемый Object"""
    def __init__(self) -> None:
        """Constructor."""
        self.observers = [] # инициализация списка наблюдателей

    def register(self, observer: Observer) -> None:
        """Регистрация нового наблюдателя на подписку"""
        self.observers.append(observer)

    def notify_observers(self, message: str) -> None:
        """Передача сообщения всем наблюдателям,
        подписанным на события данного объекта наблюдаемого класса"""
        for observer in self.observers:
            observer.update(message)
#########################
class Newspaper(Observable):
    """Газета, за новостями в которой следят тысячи людей"""
    def add_news(self, news: str) -> None:
        """Выпуск очередной новости"""
        self.notify_observers(news)

##########################################
class Observer(metaclass=ABCMeta):
    """Interface Абстрактный наблюдатель"""
    @abstractmethod
    def update(self, message: str) -> None:
        """Получение нового сообщения"""
        pass
#########################
class Citizen(Observer):
    """Обычный гражданин, который любит читнуть с утра любимую газетку"""
    def __init__(self, name: str) -> None:
        """Constructor.
        :param name: имя гражданина, чтоб не спутать его с кем-то другим
        """
        self.name = name

    def update(self, message: str) -> None:
        """Получение очередной новости"""
        print('{} узнал следующее: {}'.format(self.name, message))

##########################################
if __name__ == '__main__':
    newspaper = Newspaper()                    # создаем небольшую газету
    newspaper.register(Citizen('Иван'))        # добавляем двух человек, которые
    newspaper.register(Citizen('Василий'))     # ... ее регулярно выписывают
    # ... и вбрасываем очередную газетную утку
    newspaper.add_news('Наблюдатель - поведенческий шаблон проектирования')
```

Notifiy

```
Иван узнал следующее: Наблюдатель - поведенческий шаблон проектирования
Василий узнал следующее: Наблюдатель - поведенческий шаблон проектирования
```