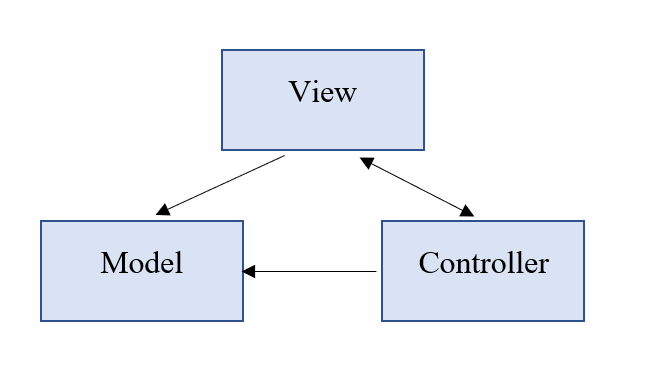
Лабораторная работа №4  
Соколов Арсений

**Just Patterns**

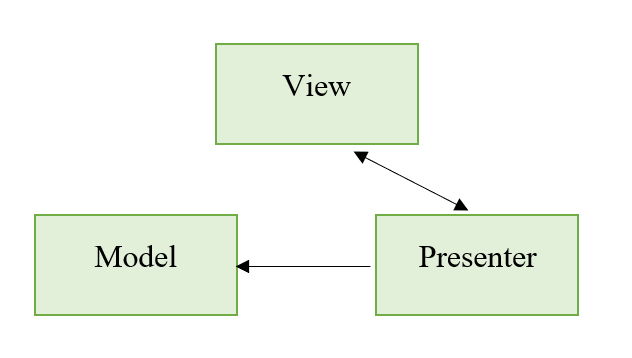
**Паттерн (шаблон) проектирования** — это продуманный способ построения исходного кода программы для решения часто возникающих в повседневном программировании проблем. Однако это не готовое решение, а просто алгоритм действий, который должен привести к желаемому результату. Паттерны проектирования разделяют на несколько групп, некоторые из них:

1. **MVC** (MVP, MVVM) - паттерны архитектуры программного обеспечения, которые определяют разделение обязанностей между графическим интерфейсом, бизнес логикой и данными.

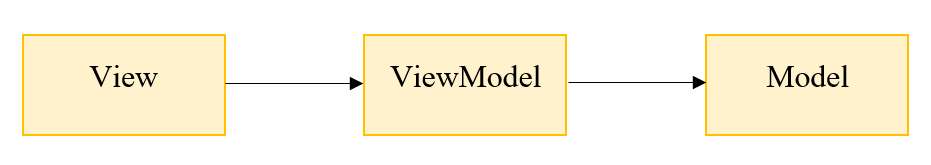
**Model-View-Controller**



**Model-View-Presenter**

****

**Model-View-View Model**

****

1. **Порождающие** **паттерны** (creational) – отвечают за процесс создания объектов в системе.
2. **Структурные** **паттерны** (structural) – определяют способы построения сложных структур из классов и объектов.
3. **Поведенческие** **паттерны** (behavioral) – описывают способы организации взаимодействия между объектами в системе.



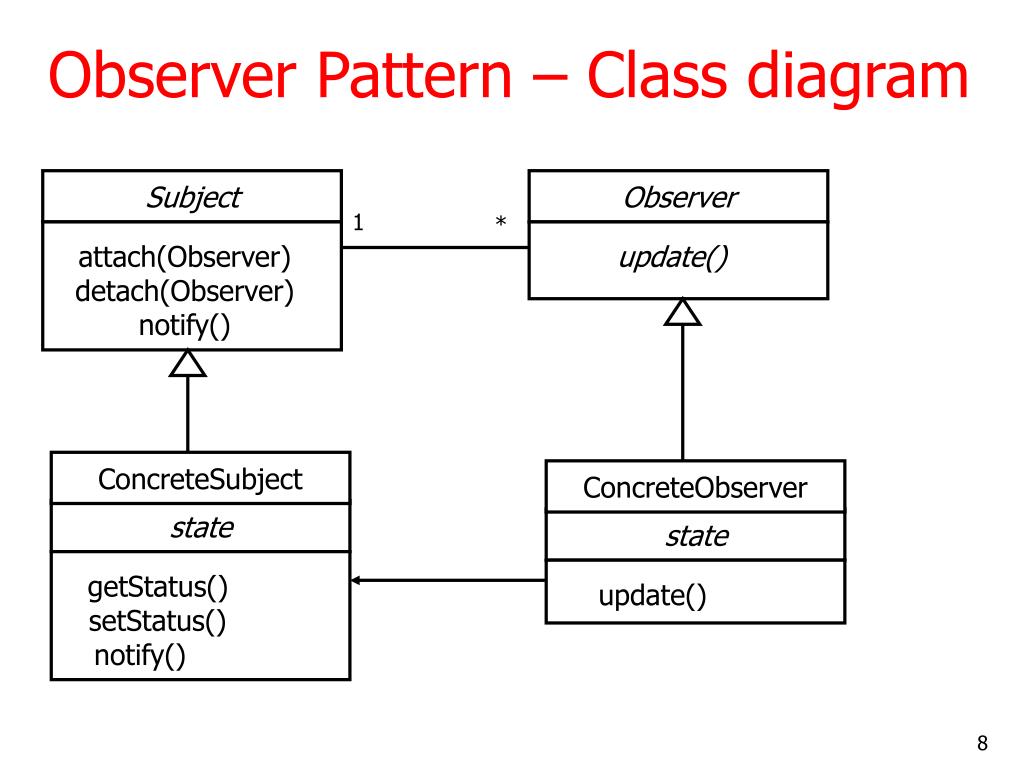
Остановимся более подробно на конкретном шаблоне.

**Наблюдатель (Observer)** – это *поведенческий* паттерн проектирования, который определяет отношение «один-ко-многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него объекты автоматически уведомляются и обновляются. Данный паттерн также называют «издатель-подписчик» (Publisher-Subscriber), поскольку отношения издателя и подписчиков характеризуют действие данного паттерна.

Применяется в случаях, когда у вас есть объект, который может изменять свое состояние, и другие объекты должны быть уведомлены об этих изменениях и реагировать на них.

Очень распространенный паттерн в реальной жизни. Как только вы (наблюдаемый объект) подписываетесь на событие (например, новая статья или сообщение), всем кто тоже подписан на это событие (наблюдателям) будет выслано уведомление.

**UML**-**диаграмма «Наблюдатель»:**

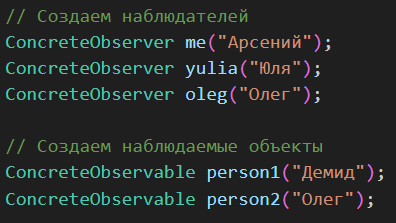


Описание классов:

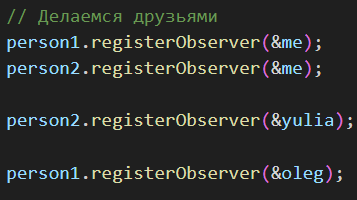
* **Observable** – интерфейс, определяющий методы для добавления, удаления и оповещения наблюдателей;
* **Observer** – интерфейс, с помощью которого наблюдатель получает оповещение;
* **ConcreteObservable** – конкретный класс, который реализует интерфейс Observable;
* **ConcreteObserver** – конкретный класс, который реализует интерфейс Observer.

**Пример реализации на языке C++**

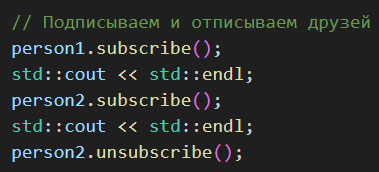
Имеем некоторых людей



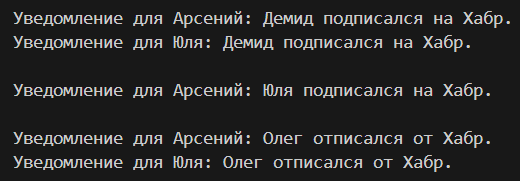
Имитируем их знакомство (наличие друг у друга в друзьях)



Пусть некоторые из них подпишутся или отпишутся на Хабр



Тогда получаем уведомления



Таким образом, уведомления о изменении статуса друга (наблюдаемый объект) получают все (наблюдатели), кто с ним дружит и только они (установлено наблюдение).

(Полный код приведен в Приложении 1)

**ПРИЛОЖЕНИЕ 1**

***Реализация шаблона «Наблюдатель» на примере уведомлений***

#include <iostream>

#include <vector>

class Observer {

public:

    virtual void update(std::string message) = 0;

};

class Observable {

protected:

    std::vector<Observer\*> observers;

public:

    void registerObserver(Observer\* observer) {

        observers.push\_back(observer);

    }

    void notifyObservers(std::string message) {

        for (Observer\* observer : observers) {

            observer->update(message);

        }

    }

};

class ConcreteObservable : public Observable {

private:

    std::string friendName;

public:

    ConcreteObservable(std::string name) : friendName(name) {}

    void subscribe() {

        notifyObservers(friendName + " подписался на Хабр.");

    }

    void unsubscribe() {

        notifyObservers(friendName + " отписался от Хабр.");

    }

};

class ConcreteObserver : public Observer {

private:

    std::string observerName;

public:

    ConcreteObserver(std::string name) : observerName(name) {}

    void update(std::string message) override {

        std::cout << "Уведомление для " << observerName << ": " << message << std::endl;

    }

};

int main() {

    // Создаем наблюдателей

    ConcreteObserver me("Арсений");

    ConcreteObserver yulia("Юля");

    ConcreteObserver oleg("Олег");

    // Создаем наблюдаемые объекты

    ConcreteObservable person1("Демид");

    ConcreteObservable person2("Олег");

    // Делаемся друзьями

    person1.registerObserver(&me);

    person2.registerObserver(&me);

    person2.registerObserver(&yulia);

    person1.registerObserver(&oleg);

    // Подписываем и отписываем друзей

    person1.subscribe();

    std::cout << std::endl;

    person2.subscribe();

    std::cout << std::endl;

    person2.unsubscribe();

    return 0;

}