Паттерн «Наблюдатель» (Observer Pattern)

Принцип работы

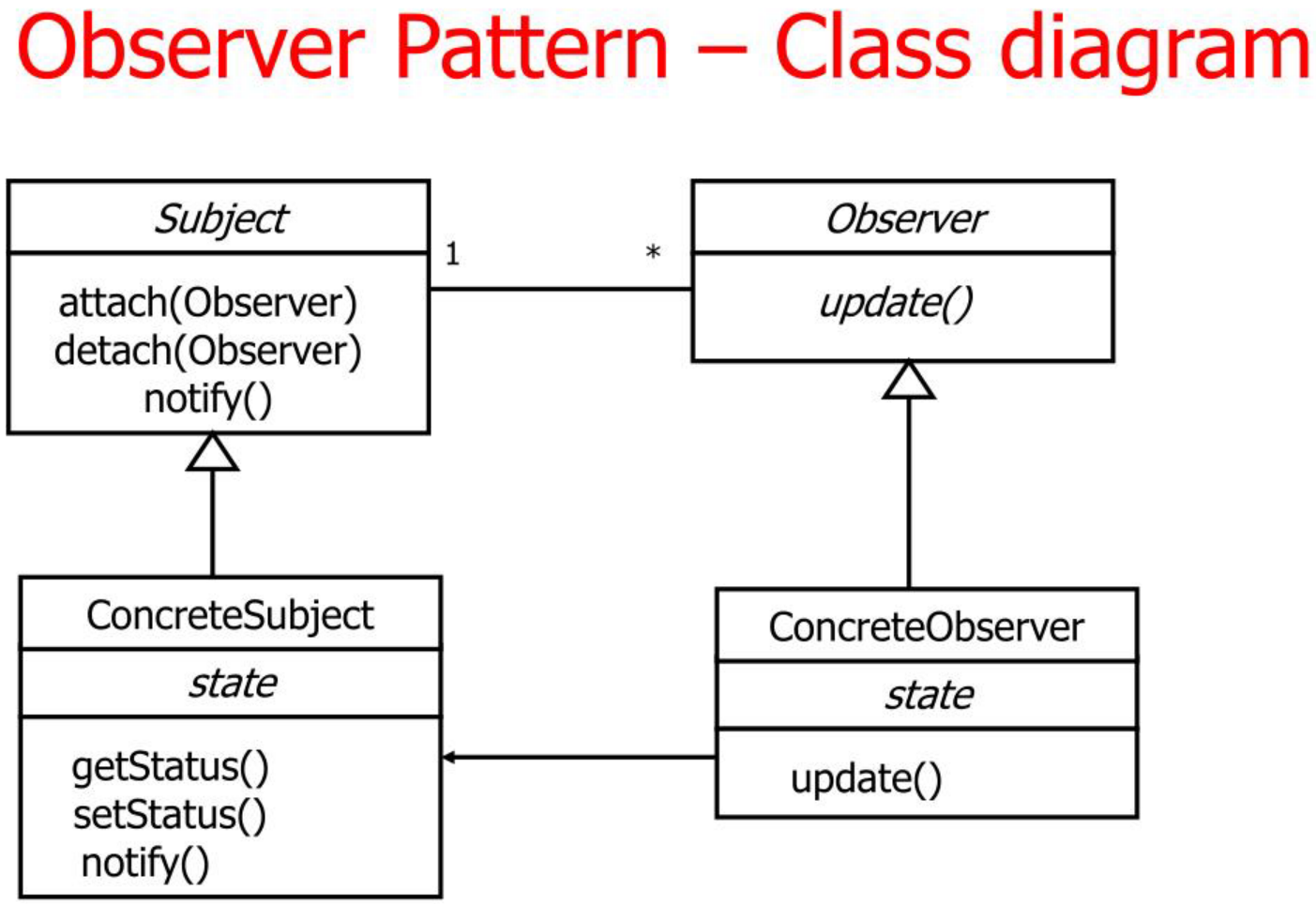
При изменении состояния одного объекта (субъекта) все зарегистрированные объекты (наблюдатели) уведомляются и обновляются автоматически. Это позволяет:

- **Избежать сильного зацепления** между компонентами — авторы субъекта и наблюдателей не должны знать много о друг друге, они взаимодействуют через интерфейс.
- **Динамические отношения** — наблюдатели можно добавлять или удалять во время работы приложения.

```cpp
#include <iostream>

// Устройства, которые показывают температуру
class Phone {
public:
    void show(int temp) {
        std::cout << "Телефон: Температура " << temp << " °C\n";
    }
};

class Watch {
public:
    void show(int temp) {
        std::cout << "Часы: Сейчас " << temp << " °C\n";
    }
};

// Погодная станция — ЗНАЕТ о каждом устройстве напрямую
class WeatherStation {
private:
    int temperature;
    Phone phone;   // ← Жёстко встроены!
    Watch watch;   // ← Нельзя убрать

public:
    void setTemperature(int t) {
        temperature = t;
        // Каждое устройство вызывается ВРУЧНУЮ
        phone.show(temperature);
        watch.show(temperature);
        // 🔥 Если добавить новое устройство — придётся ЛЕЗТЬ СЮДА!
    }
};

int main() {
    setlocale(LC_ALL, "rus"); // Единственная строка для русского — без изменений

    WeatherStation weather;
    weather.setTemperature(25);

    return 0;
}
```

Паттерн «Наблюдатель»: Структура

UML – общая структура



Паттерн «Наблюдатель»: пример использования

```cpp
#include <iostream>
#include <vector>

// Кто получает уведомления
class Observer {
public:
    virtual ~Observer() = default;           // Важно: виртуальный деструктор!
    virtual void update(int temp) = 0;
};

// Объект, за которым наблюдают
class Weather {
private:
    std::vector<Observer*> observers;
    int temperature;

public:
    void setTemp(int t) {
        temperature = t;
        for (auto* o : observers) {
            o->update(temperature);  // Оповещаем всех
        }
    }

    void addObserver(Observer* o) {
        observers.push_back(o);
    }
};
```

```cpp
// Один из наблюдателей
class Phone : public Observer {
public:
    void update(int temp) override {
        std::cout << "Телефон: " << temp << "°C\n";
    }
};

class Watch : public Observer {
public:
    void update(int temp) override {
        std::cout << "Часы: " << temp << "°C\n";
    }
};

// 🚀 Главное — добавь main(), иначе компилятор не знает, с чего начать!
int main() {
    setlocale(LC_ALL, "rus");
    Weather weather;
    Phone phone;
    Watch watch;

    weather.addObserver(&phone);
    weather.addObserver(&watch);

    weather.setTemp(28);  // Должно вывести два сообщения

    return 0;
}
```