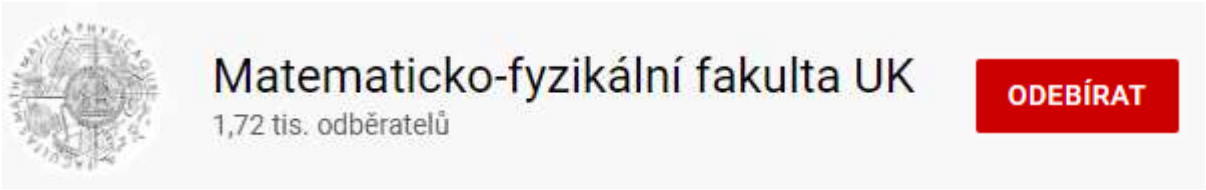
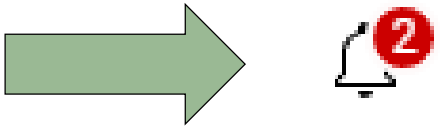
Observer

Motivace

Nové video



Notifikace



Změna teploty

Chceme vědět, že se změnila teplota

Komunikace – naivní řešení

Změnila se teplota?

Ano, na 15°C

Změnila se teplota?

Ne

...

Změnila se teplota?

Ne

Změnila se teplota?

Ano, na 12°C

Změnila se teplota?

Ne

Komunikace – řešení pomocí návrhového vzoru OBSERVER – neformálně

Stanice, notifikuj mě o změně teploty

Teplota se změnila na 15°C

Teplota se změnila na 12°C

Už mě nenotifikuj

Komunikace – řešení pomocí návrhového vzoru OBSERVER – formálně

Subject (vydavatel)

`Subject.Attach(Observer);`

`Observer.Update(15°C);`

`Observer.Update(12°C);`

`Subject.Detach(Observer);`

Observer (předplatitel)

Observer – definice

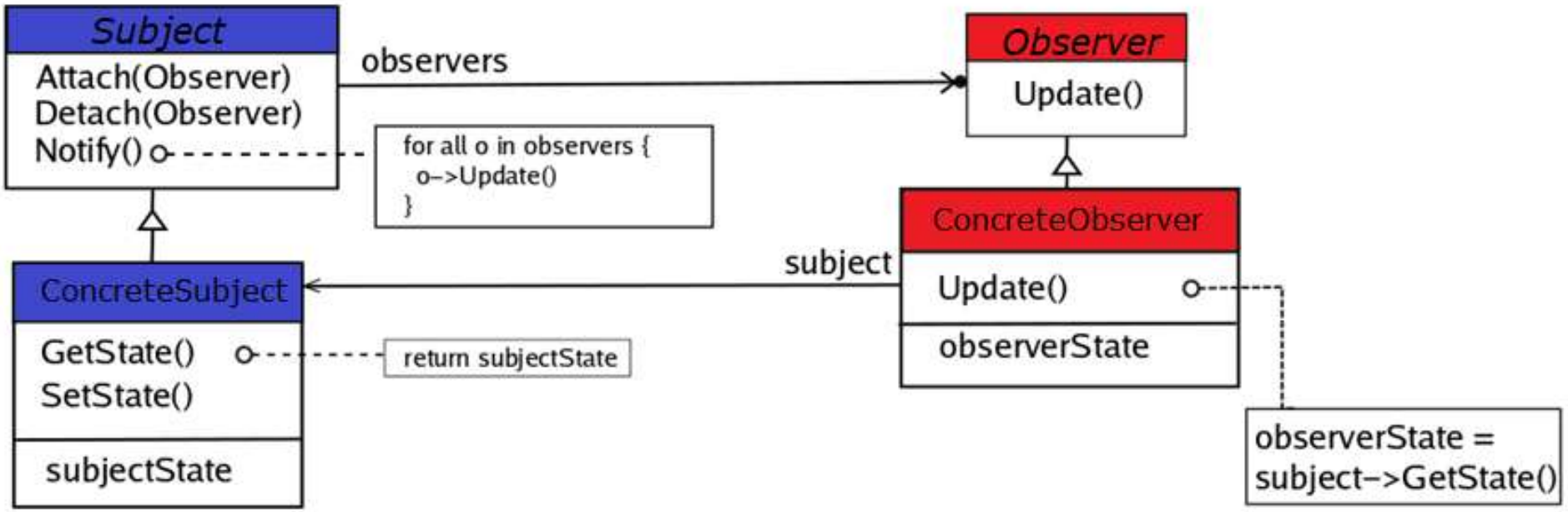
- behaviorální návrhový vzor **Observer**
  - definuje vazbu 1 : N (Subjekt : Observer)
  - při změně stavu Subjektu jsou všechny na něm závislé Observery automaticky notifikovány a aktualizovány

1
Subject

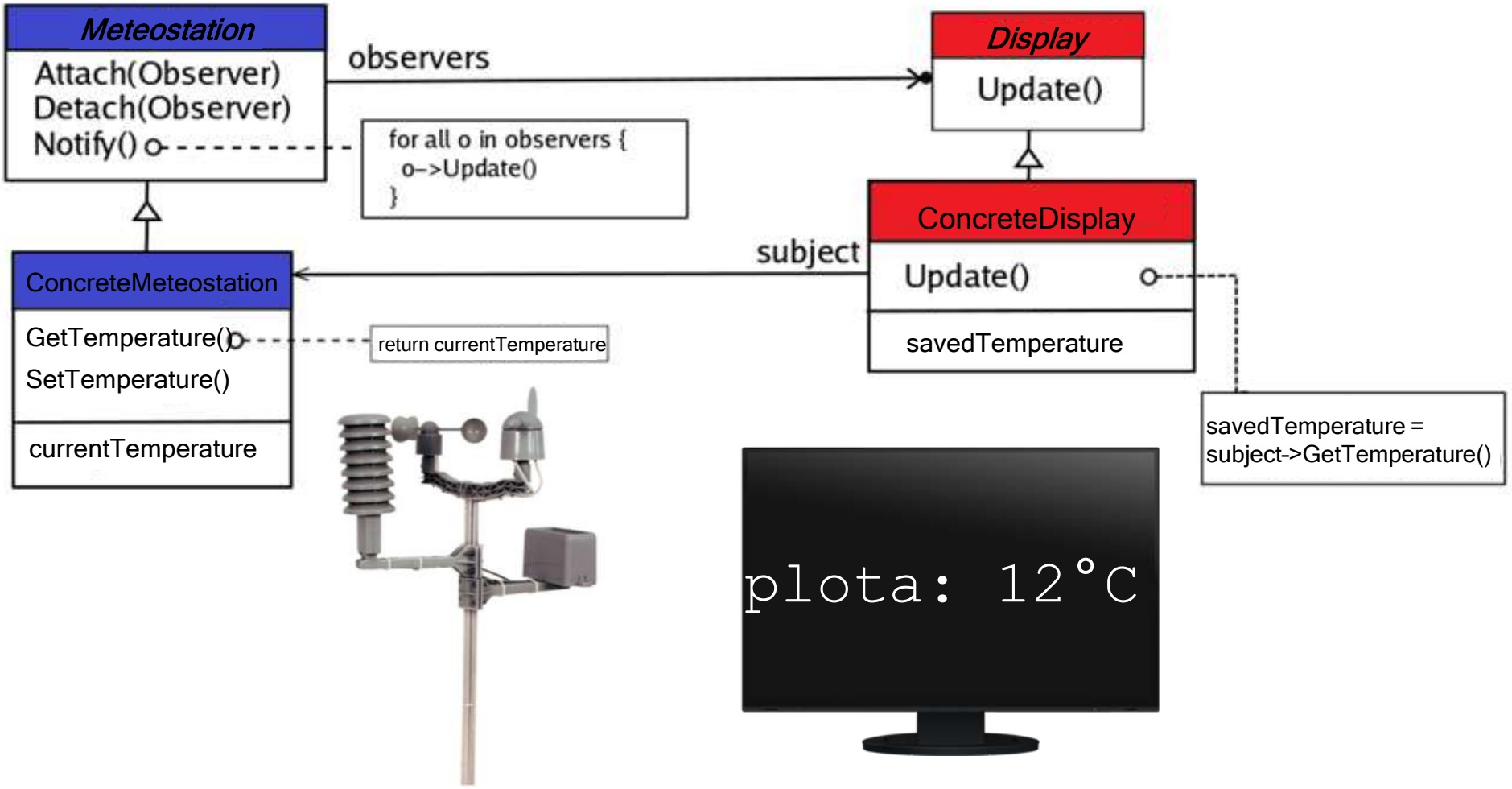
N Observers


```
for(observer :
attached_observers)
    observer.Update(12°C);
```

Observer – UML class diagram

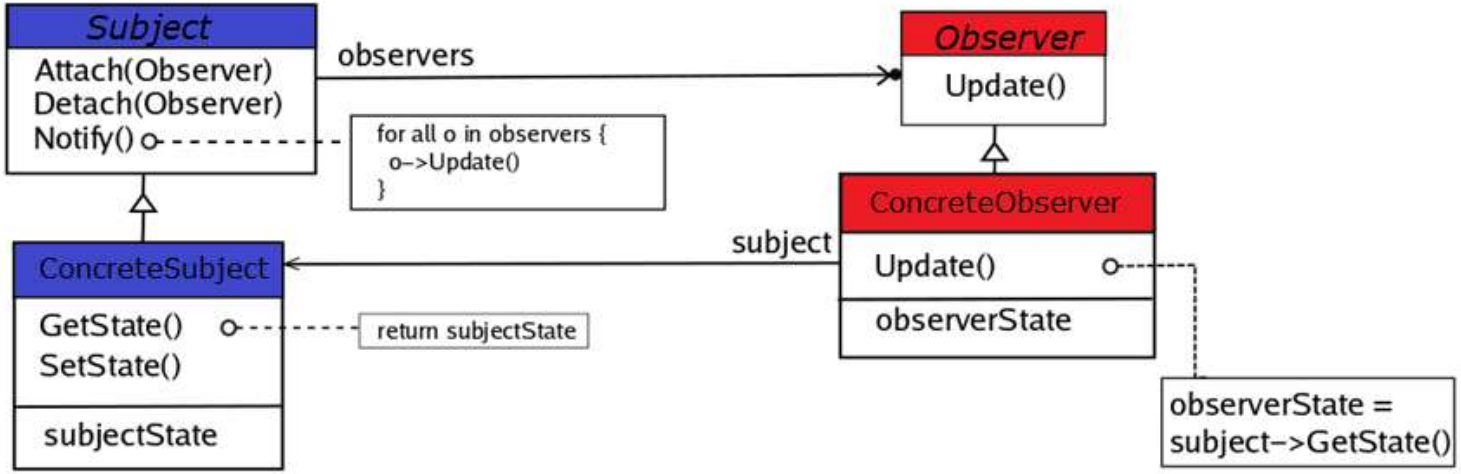


Příklad – UML class diagram



Pozorování z UML class diagramu

- rozdělení zodpovědnosti
- abstract coupling
  - opětovná použitelnost
- broadcast komunikace
- nepředpověditelné trvání Update



DETAILY IMPLEMENTACE

ANEB VARIANTY OBSERVERU, JEJICH VLASTNOSTI A (NE)VÝHODY

Detaily implementace
Efektivita update

- problém
  - notifikujeme každý observer, že nastala změna
  - mnoho z nich to nemusí zajímat

- řešení
  - specifikovat [redacted] mají
  - potřebujeme [redacted]
  - rozšíření **Ob**[redacted]

ne obrázek!

```
// Observer, které
void Display::Update(string change) {
    if (change == "temperature") {
        observed_state.temperature = observed_subject->GetState().temperature;
    }
    else {
        // not interested
    }
}
```

Detaily implementace
Efektivita update

- řešení
  - rozšíření **Subject::Attach()**

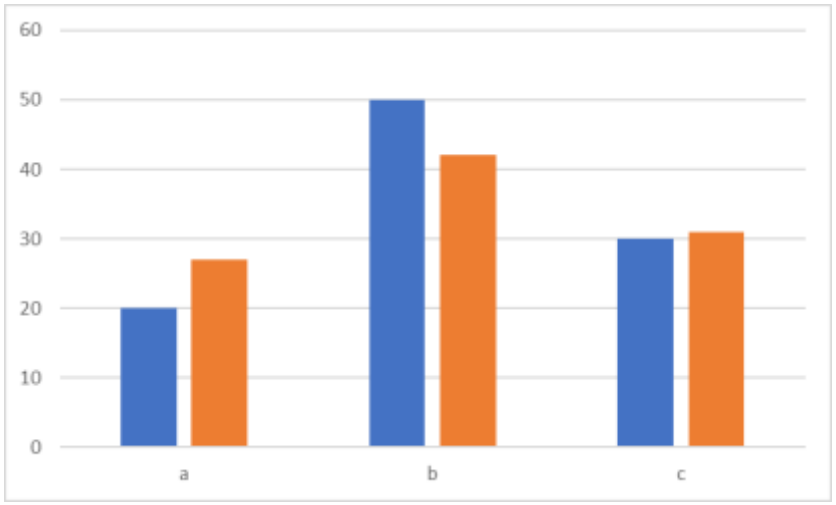
```
#define OBSERVER observer.first
#define INTEREST observer.second

void Meteostation::Attach(Observer* observer, string interested_in) {
    attached_observers.push_back(make_pair(observer, interested_in));
}

void Meteostation::Notify(string what_changed) {
    for (auto&& observer : attached_observers) {
        // notifikujeme, jen když to daný observer zajímá
        if (INTEREST == what_changed) {
            OBSERVER->Update();
        }
    }
}
```

  - příklad

```
document.getElementById("someButton").addEventListener("click", someFunction);
document.getElementById("someOtherButton").addEventListener("focus", someOtherFunction);
```

Detaily implementace
Pozorování více Subjectů

a	20
b	50
c	30



a	27
b	42
c	31

- musíme rozšířit Update() interface
  - Subjekt se předá jako argument

```
for(observer :
attached_observers)
```

Detaily implementace
Kdo a kdy volá Notify()

- **Subject** volá Notify()
  - Observer se o to nemusí starat
  - potenciálně zbytečné Update-y

Teplota: 11°C
Rychlost větru: 10m/s
Teplota: 12°C
Rychlost větru: 10m/s
Změna stavu, volání Notify()

Teplota: 12°C
Rychlost větru: 9m/s

Změna stavu, volání Notify()

Subject

Subject.SetTemperature(12 °C);

Subject.Notify();

Observer.Update();

Subject.SetWindSpeed(9m/s);

Subject.Notify();

Observer.Update();

Chceme změnit teplotu a rychlost větru Subjectu

Observer

Detaily implementace
Kdo a kdy volá Notify()

- **Observer** volá Notify()
  - žádné zbytečné Update-y
  - Observer má větší zodpovědnost → vyšší náchylnost k chybám

Teplota: 11°C
Rychlost větru: 10m/s

Teplota: 12°C
Rychlost větru: 10m/s

Teplota: 12°C
Rychlost větru: 9m/s

Subject

Subject.SetTemperature(12°C);

Subject.SetWindSpeed(9m/s);

Subject.Notify();

Observer.Update();

Chceme změnit teplotu a rychlost větru Subjectu

Observer

Detaily implementace

Push x Pull

Observers

Subject

Subject.Notify();

Observer.Update(12°C);

Observer.Update(12°C);

Detaily implementace

Push x **Pull**

Observers

Subject

Subject.Notify();

Observer.Update();

Subject.GetState();

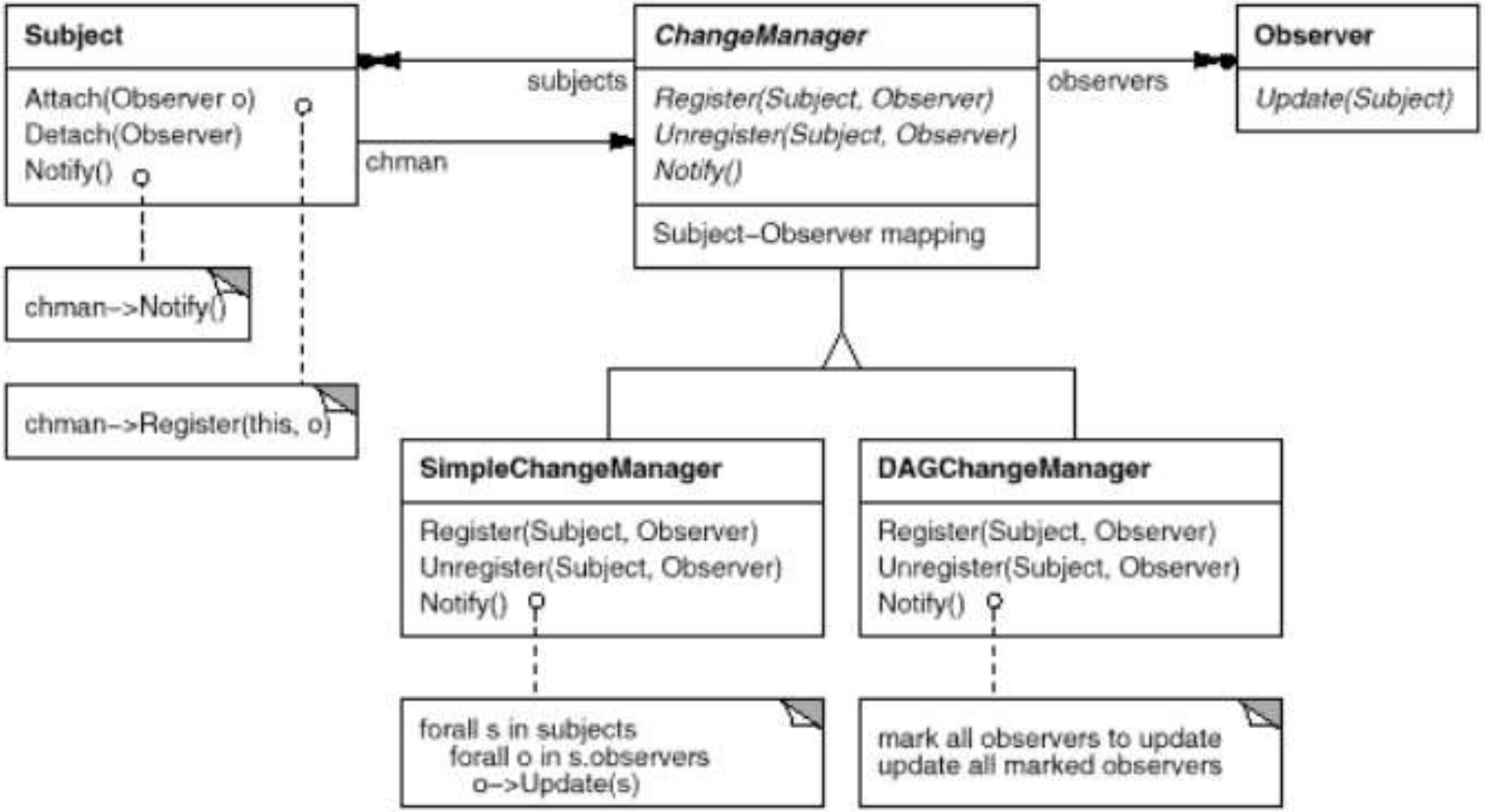
Observer.Update();

Detaily implementace
Rozšíření o Change Manager

- motivace
  - komplikované vazby Subject-Observer
  - stav jednoho Subjektu závisí na stavu jiného
  - Observer se zajímá o změnu stavu více Subjektů
- **Change Manager**
  - stará se o vazby mezi Subject-Observer
  - definuje update-ovací strategii
    - nejprve update-uje Subjecty, potom Observery
- varianty
  - SimpleChangeManager
  - DAGChangeManager

Detaily implementace

Change Manager - UML diagram



PŘÍKLAD IMPLEMENTACE

```cpp
#define OBSERVER observer.first
#define INTEREST observer.second

class Subject {
protected:
    vector<pair<Observer*, string>> attached_observers;
    Subject() {}

public:
    void Attach(Observer* observer, string interested_in) {
        observer->observed_subject = this;
        attached_observers.push_back(make_pair(observer, interested_in));
    }

    void Detach(Observer* to_deteach) {
        for (auto&& observer : attached_observers) {
            if (OBSERVER == to_deteach) {
                to_deteach->observed_subject = nullptr;
                attached_observers.erase(observer);
            }
        }
    }

    void Notify(std::string what_changed) {
        for (auto&& observer : attached_observers) {
            if (INTEREST == what_changed) {
                OBSERVER->Update();
            }
        }
    }
};
```

```cpp
class Observer {
protected:
    Observer() {}

public:
    Subject* observed_subject = nullptr;
    virtual void Update() = 0;
    virtual ~Observer() {}
};
```

```
class Meteostation : public Subject {
private:
    Meteostation_state current_state;
public:
    Meteostation() {
        current_state.temperature = 0;
        current_state.wind_speed = 0;
    }

    Meteostation_state GetState() {
        return current_state;
    }

    void SetState(Meteostation_state new_state) {
        bool temperature_change = new_state.temperature != current_state.temperature;
        bool wind_speed_change = new_state.wind_speed != current_state.wind_speed;

        current_state = new_state;

        if (temperature_change && !wind_speed_change) {
            Notify("all");
            Notify("temperature");
        } else if (!temperature_change && wind_speed_change) {
            Notify("all");
            Notify("wind_speed");
        } else if (temperature_change && wind_speed_change) {
            Notify("all");
            Notify("wind_speed");
            Notify("temperature");
        }
    }
};

struct Meteostation_state {
    int temperature;
    int wind_speed;
};
```

```
class Display : public Observer {
public:
    Meteostation_state saved_state;

    Display() {}

    void Update() {
        saved_state = observed_subject->GetState();
    }
};
```

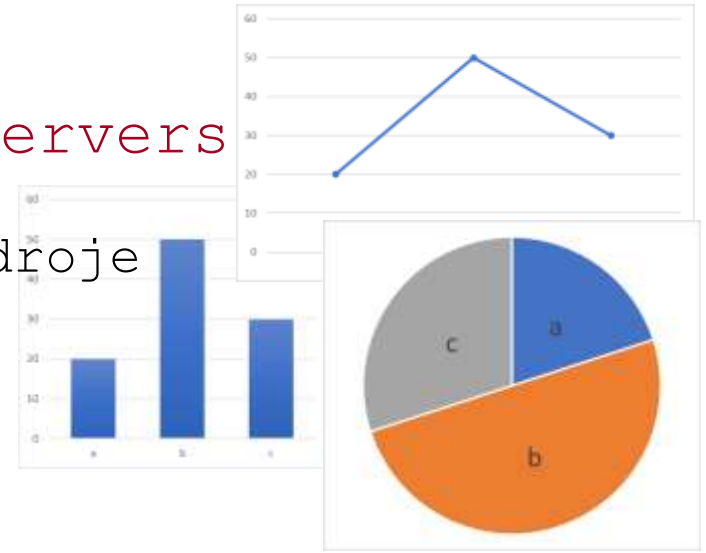
```
int main() {
    Meteostation ms;
    Meteostation_state state;

    Display d1;
    Display d2;

    ms->Attach(d1, "all");
    ms->Attach(d2, "temperature");

    state.temperature = 12;
    state.wind_speed = 21;
    ms->SetState(state);

    ms->Detach(d2);
}
```

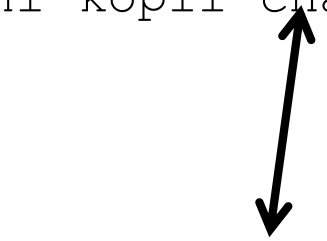
Na co si dávat pozor

- cykly
  - dva Observery se navzájem sledují → zacyklení
- problémy s pamětí
  - vymazání Observeru, který je závislý na Subjektu
  - vymazání Subjektu, který je pozorován Observerem
- pořadí update-ů Observerů
  - není garantované

Kde se Observer používá

- aplikace s GUI
  - event listenery
  - různé zobrazení dat z jednoho zdroje
- klient-server aplikace
  - skupinový chat
    - Subjekt … chat na serveru
    - Observer … konkrétní osoby (s lokální kopií chatu)
  - YouTube notifikace
    - Subjekt … kanál
    - Observer … odběratelé
- konkrétní využití
  - JavaScript event listenery
  - Obecné GUI knihovny
    - WxWidgets (C++)



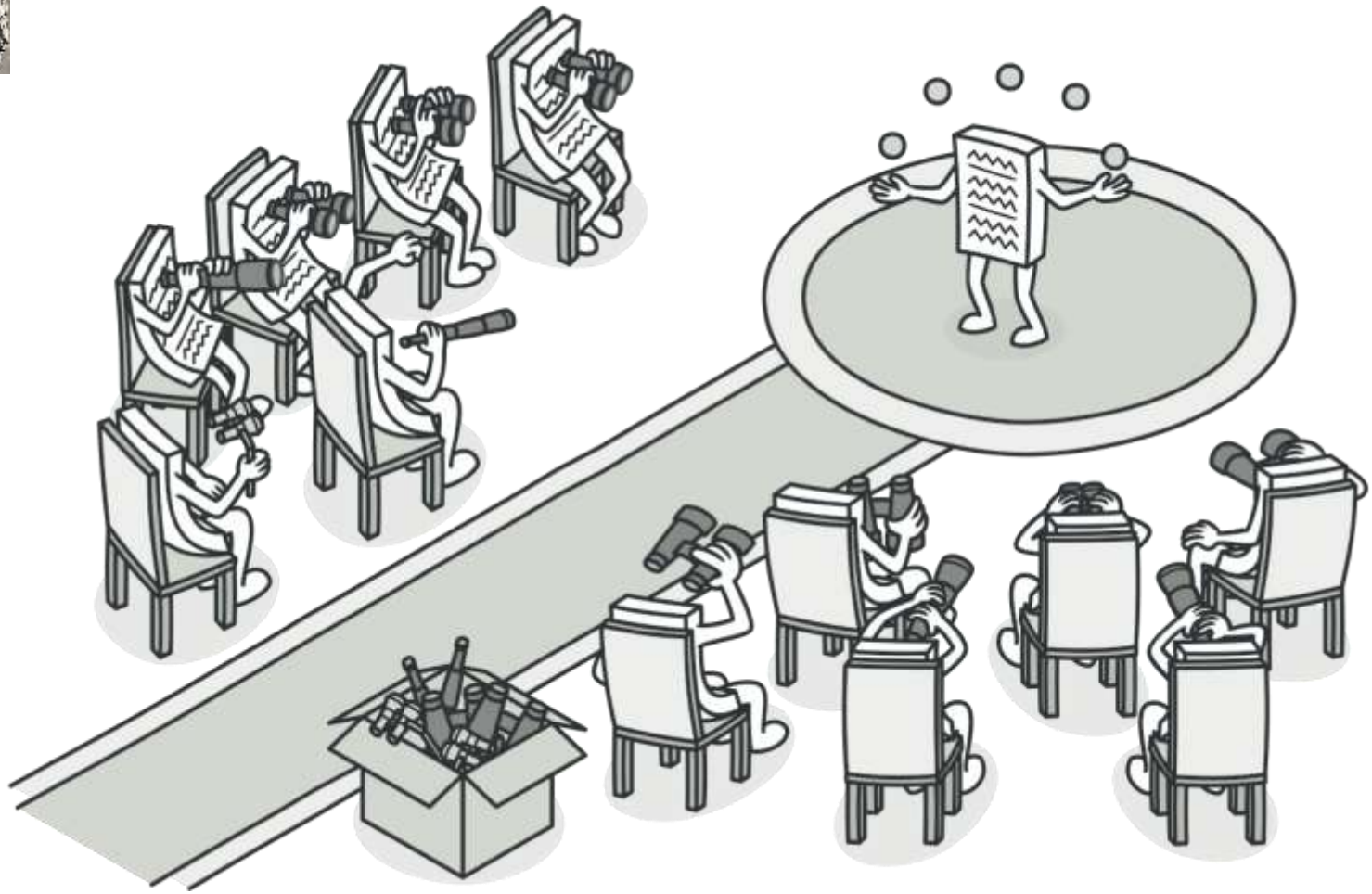


a	20
b	50
c	30

Subject

Související návrhové vzory

- **Mediator**
  - Tím, že Change Manager schovává komplexní update-ovací strategii, se chová jako Mediator.
- **Singleton**
  - Change Manager může být navržen jako Singleton kvůli tomu, aby byl unikátní a globálně přístupný.



DĚKUJI ZA POZORNOST