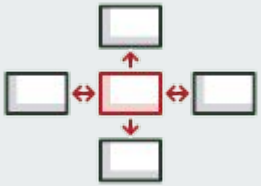
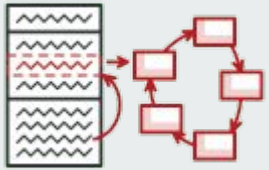
Návrhové vzory: Behavioral patterns

(Mediator, Observer, State, Memento)

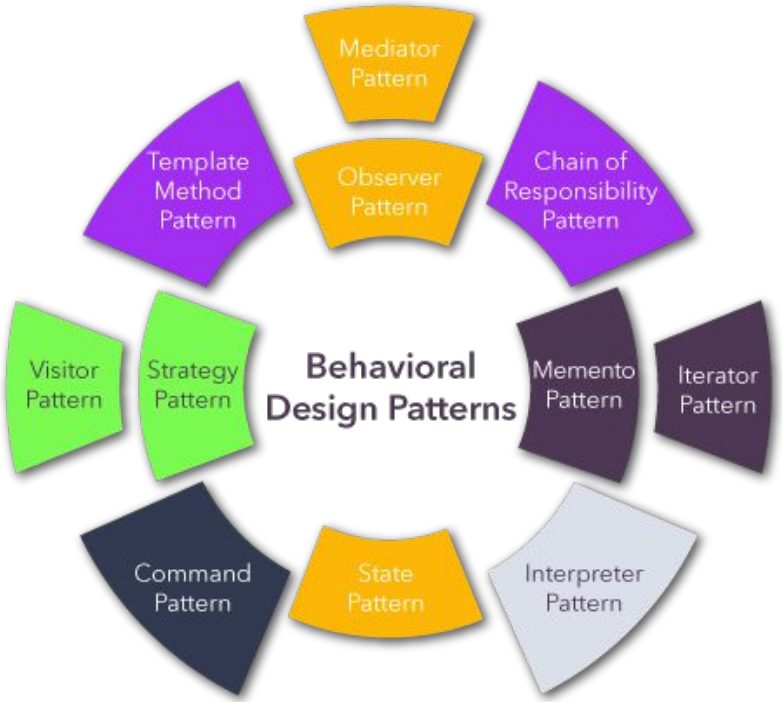





Tomáš Popík, Matej Peluha, Ľuboš Sremaňák, Katarína Popíková

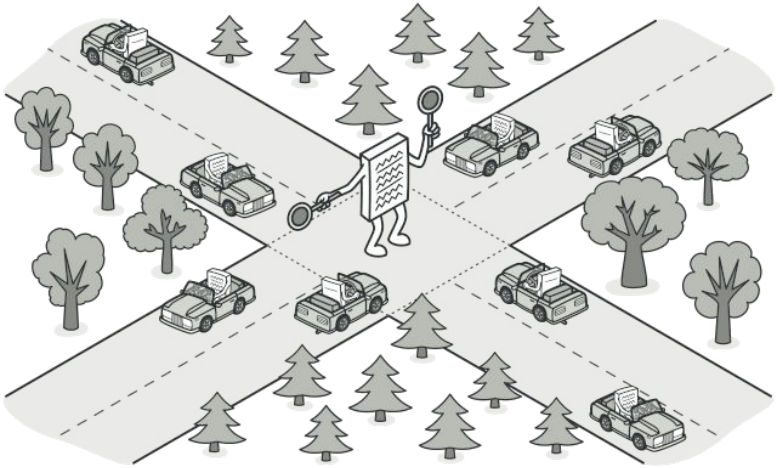
Behavioral patterns

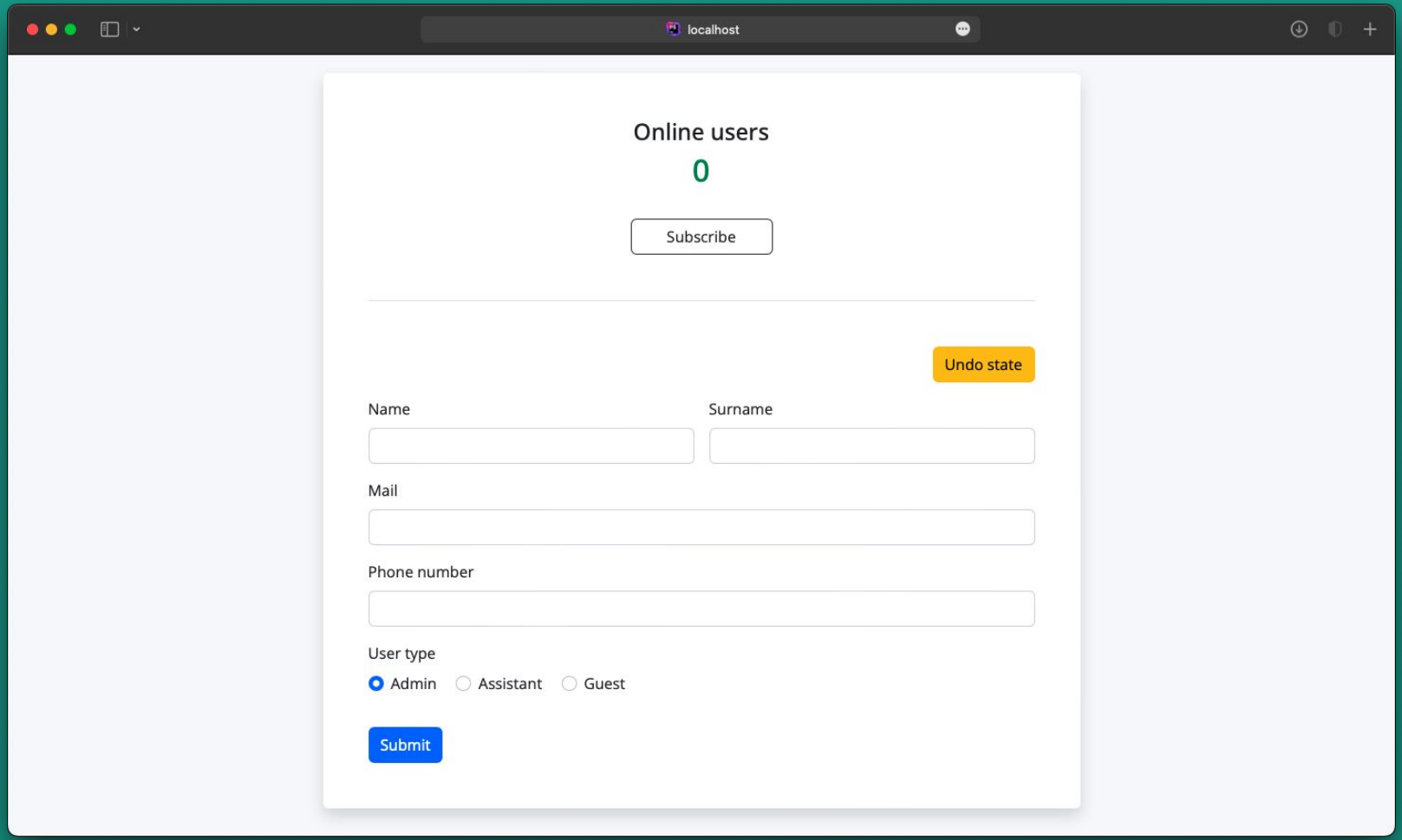
- komunikácia medzi objektami
- flexibilita
- efektivita
- enkapsulácia



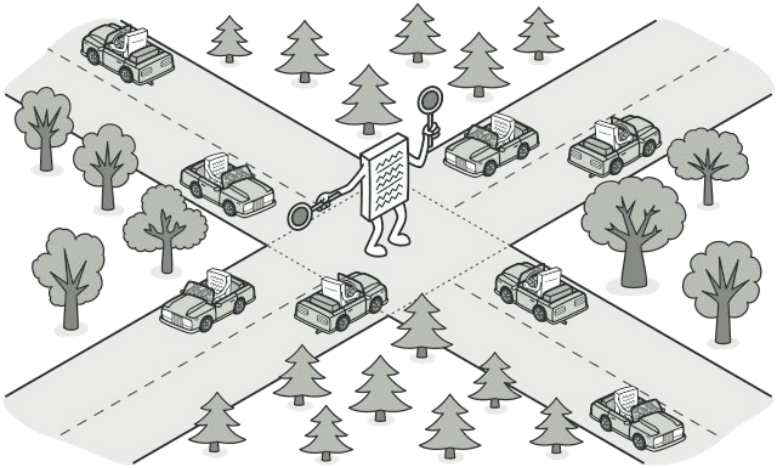
Source: https://data-flair.training/blogs/design-patterns-in-java/

Aplikácia



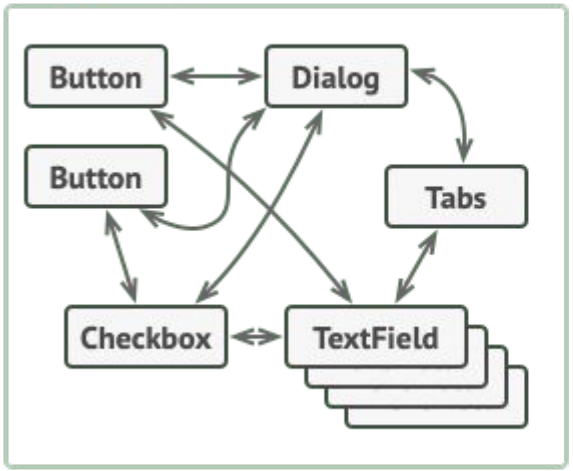

localhost
Online users
0
Subscribe
Undo state
Name
Surname
Mail
Phone number
User type
Admin
Assistant
Guest
Submit

Mediator

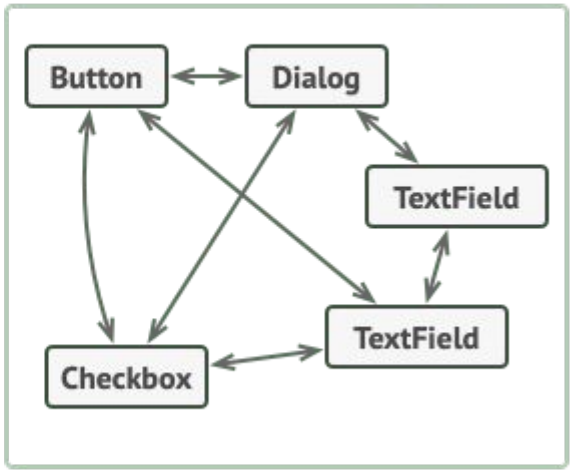


Problém

Profile Dialog

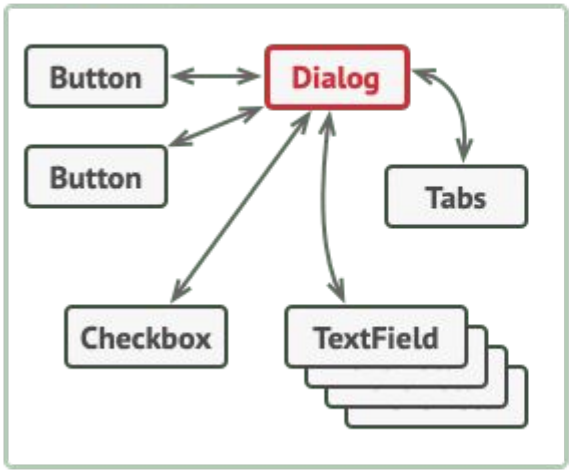


LogIn Dialog

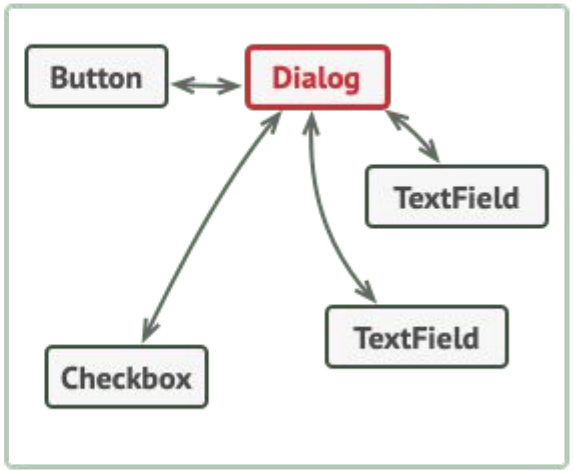


Riešenie

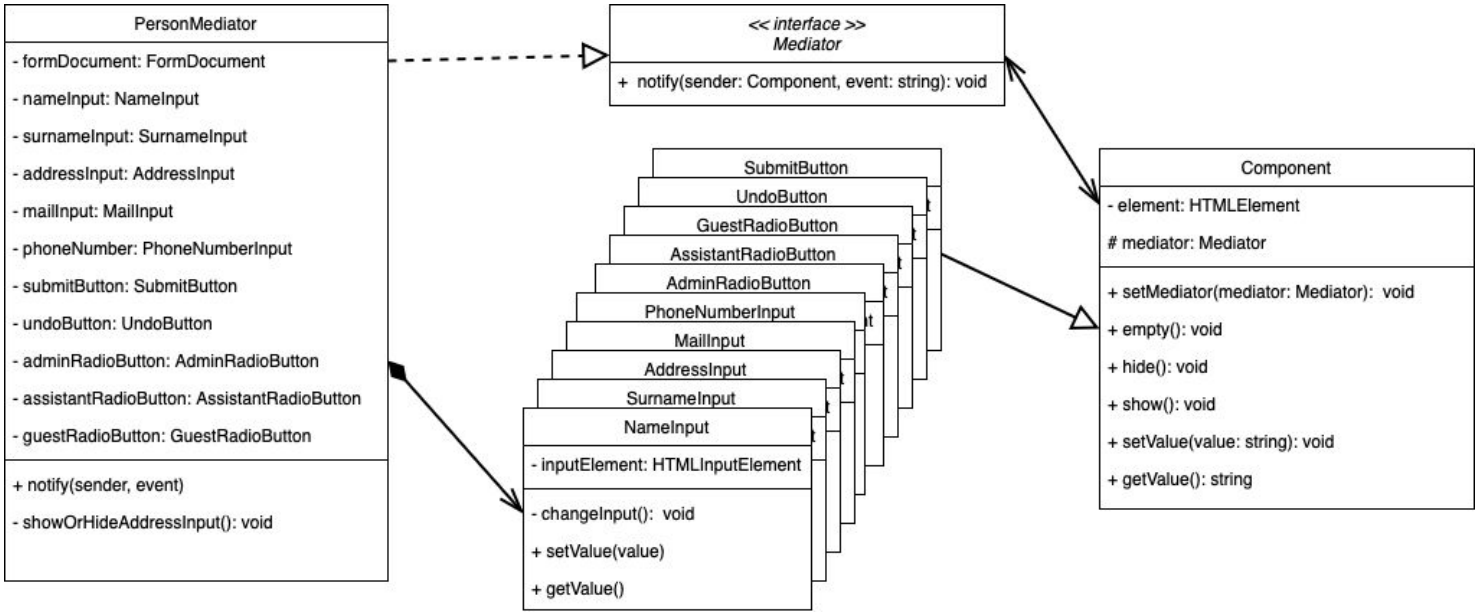
Profile Dialog



LogIn Dialog



Implementácia



Implementácia

```
const undoButton = new UndoButton(renderer.get('undo-button') as HTMLInputElement);
const nameInput = new NameInput(renderer.get('name') as HTMLInputElement);
const surnameInput = new SurnameInput(renderer.get('surname') as HTMLInputElement);
const mailInput = new MailInput(renderer.get('mail') as HTMLInputElement);
const phoneNumberInput = new PhoneNumberInput(renderer.get('phone-number') as HTMLInputElement);
const adminRadioButton = new AdminRadioButton(renderer.get('admin-radio-option') as HTMLInputElement);
const assistantRadioButton = new AssistantRadioButton(renderer.get('assistant-radio-option') as HTMLInputElement);
const guestRadioButton = new GuestRadioButton(renderer.get('guest-radio-option') as HTMLInputElement);
const addressInput = new AddressInput(renderer.get('address') as HTMLInputElement);
const submitButton = new SubmitButton(renderer.get('submit-button') as HTMLInputElement);

const mediator = new PersonMediator(
    undoButton,
    nameInput,
    surnameInput,
    mailInput,
    phoneNumberInput,
    adminRadioButton,
    assistantRadioButton,
    guestRadioButton,
    addressInput,
    submitButton
);
```

Výhody a nevýhody

+ Single responsibility principle

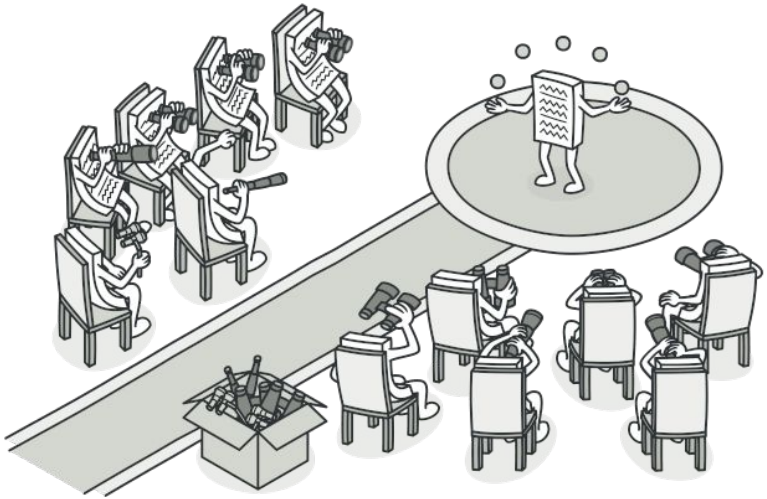
+ Open/Closed principle

+ Obmedzíme priame prepojenia medzi komponentami

+ Jednoduchšie prepoužitie komponentov

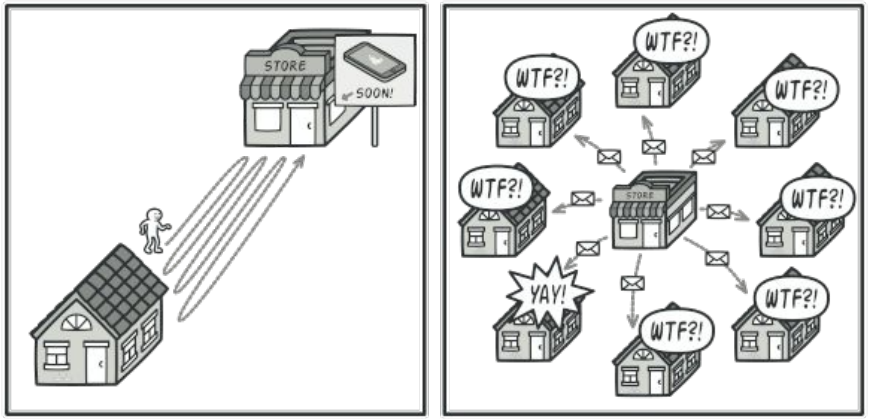
- God object

Observer



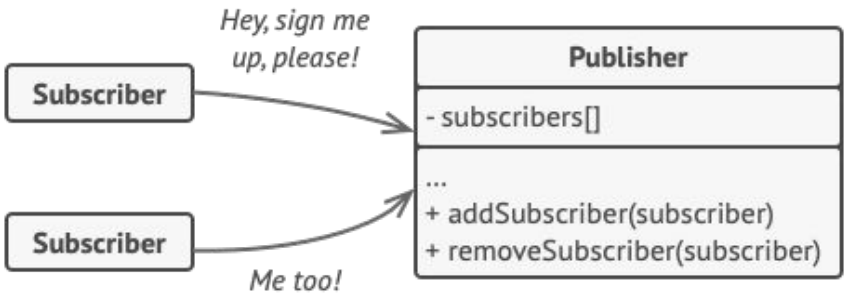
Problém

- Obchod a Zákazník

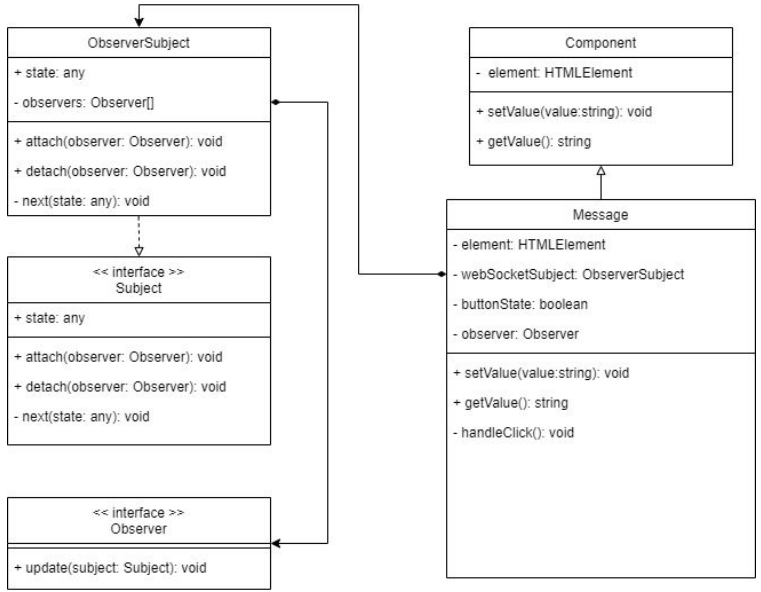


Riešenie

- Subject/Publisher & Subscribers



Implementácia



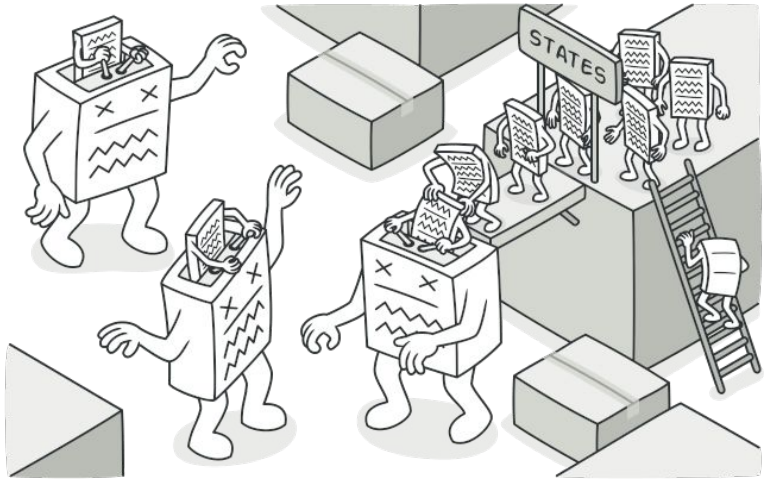
Výhody a nevýhody

+ Nezávislosť kódu Subscribera na kóde Publishera

+ Vzťahy medzi objektmi, je možné stanoviť za behu

- Náhodné poradie oznamovania udalosti

State

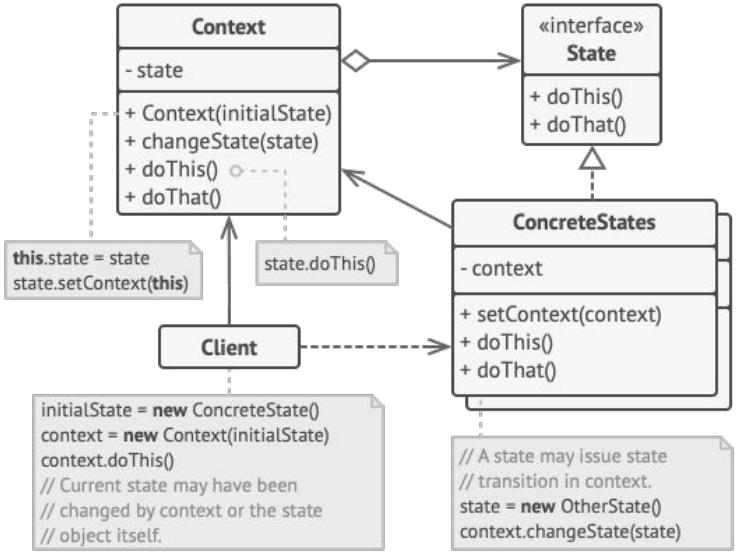


Problém - state machine

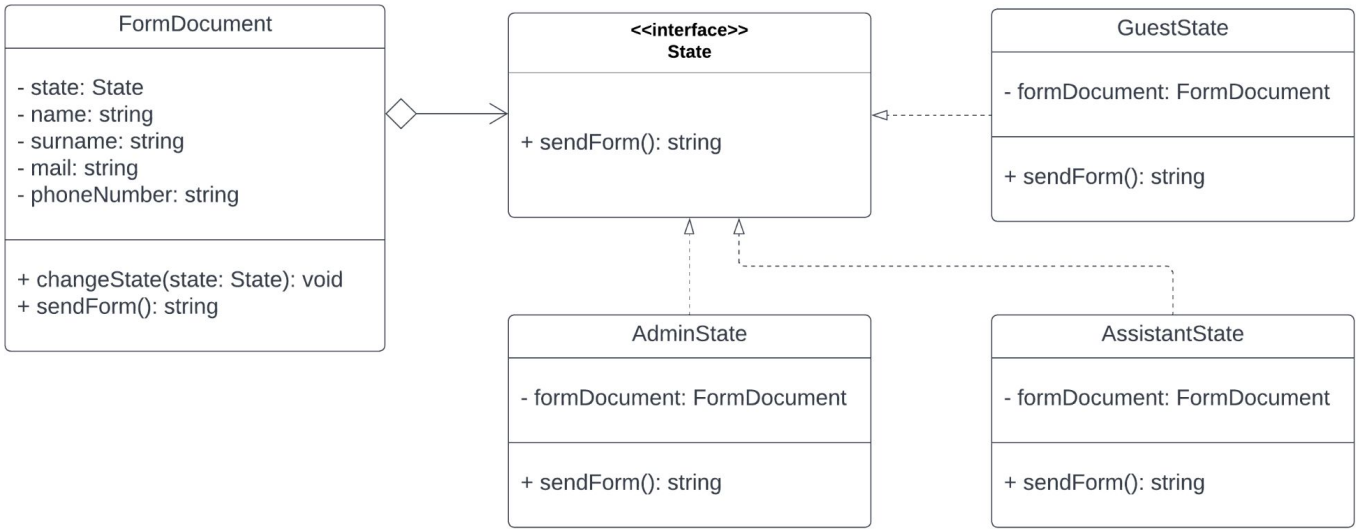
```
class FormDocument is
    field state: string
    // ...
    method publish() is
        switch (state)
            "admin":
                saveFormInDatabase()
                break
            "assistant":

                    sendFormToAdmin()
                break
            "guest":
                sendFormForAutomaticTests()
                break
```

Riešenie - state pattern



Implementácia



Výhody a nevýhody

+ Single responsibility principle

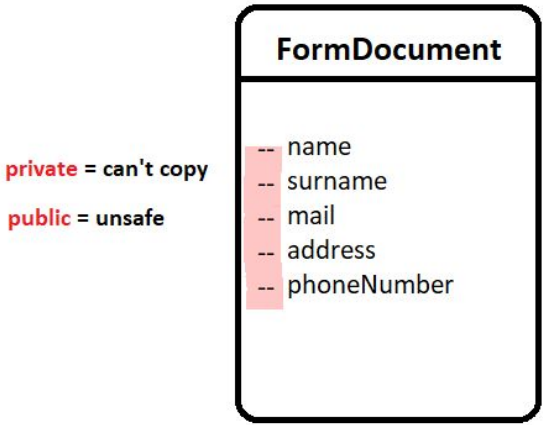
+ Open/Closed principle

+ Zjednodušuje kód contextu

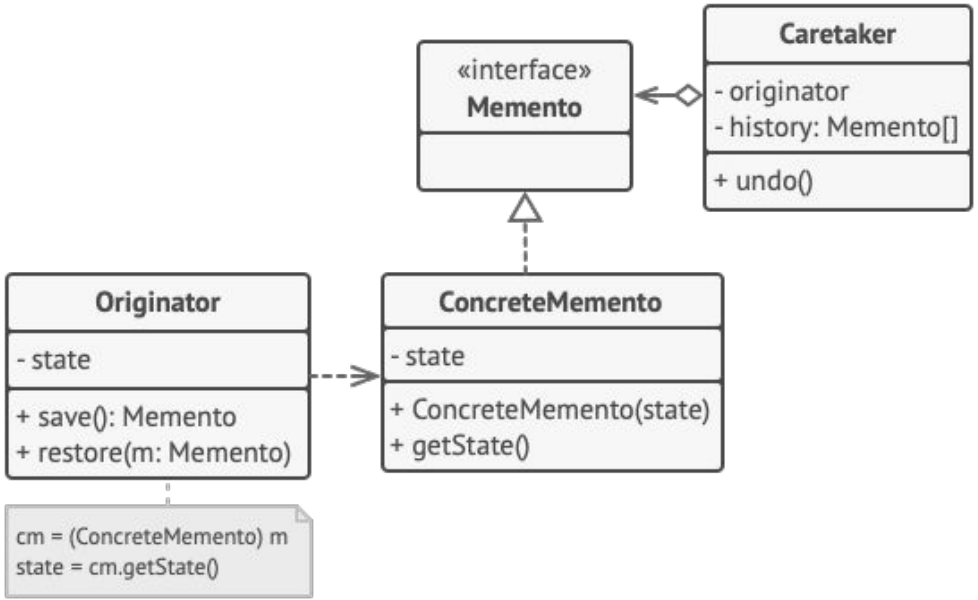
- Overkill pri malom počte stavov

Memento

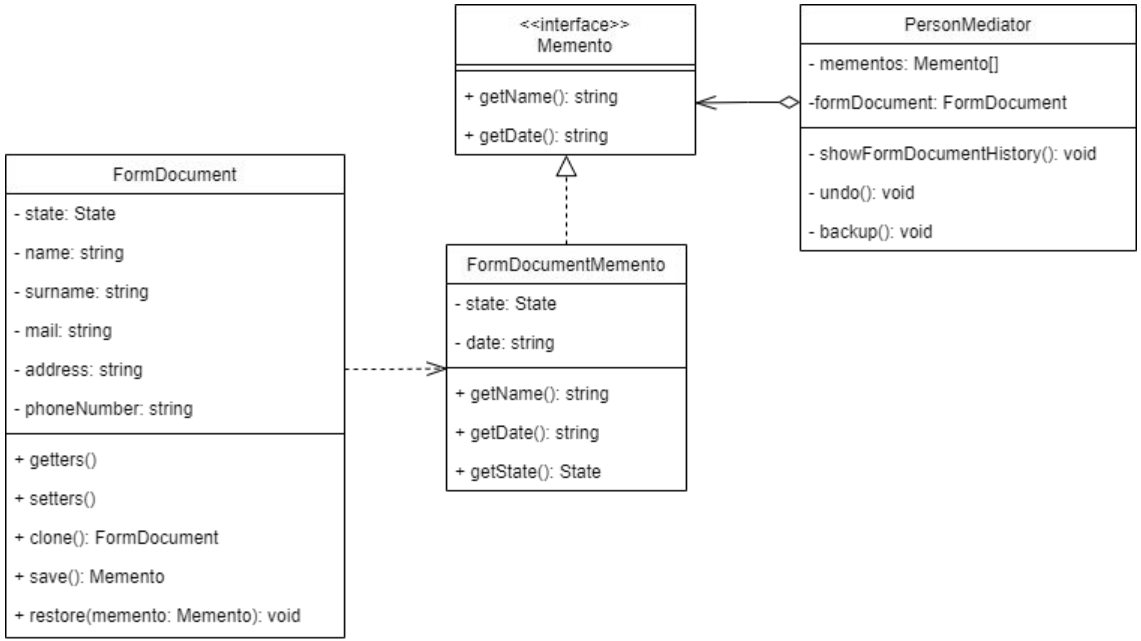
Problém


private = can't copy
public = unsafe
FormDocument
-- name
-- surname
-- mail
-- address
-- phoneNumber

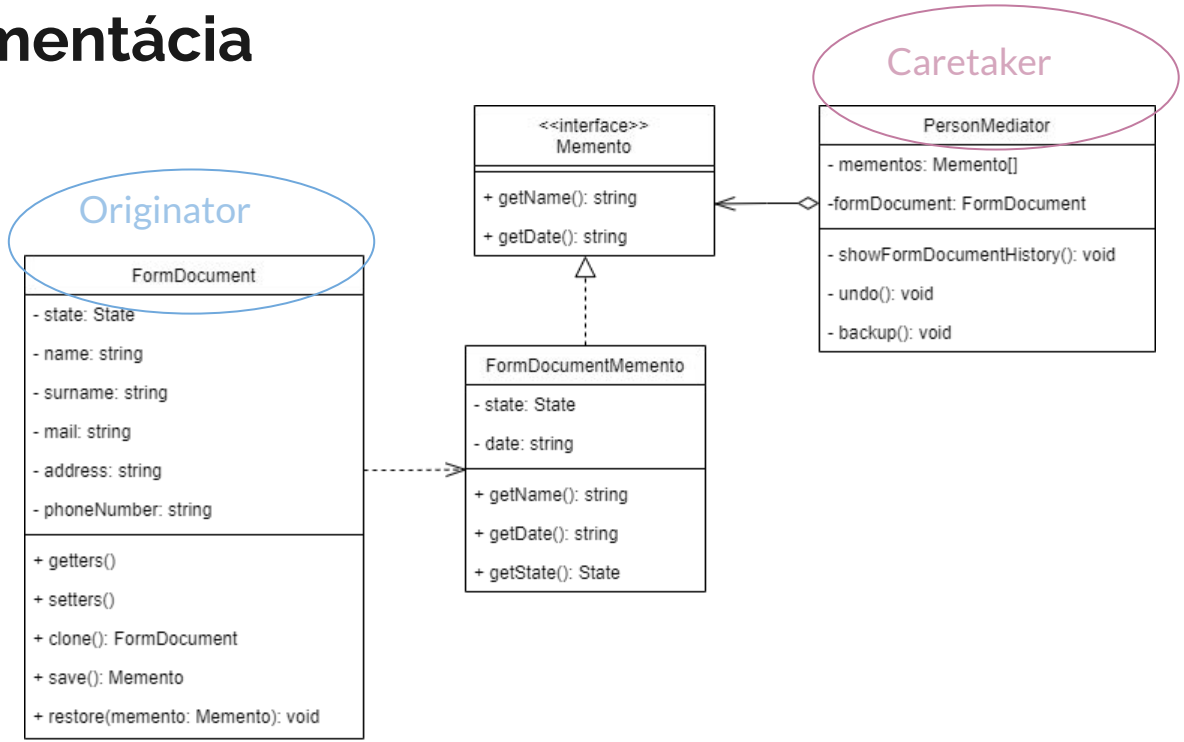
Riešenie


«interface»
Memento
Caretaker
- originator
- history: Memento[]
+ undo()
Originator
- state
+ save(): Memento
+ restore(m: Memento)
ConcreteMemento
- state
+ ConcreteMemento(state)
+ getState()
cm = (ConcreteMemento) m
state = cm.getState()

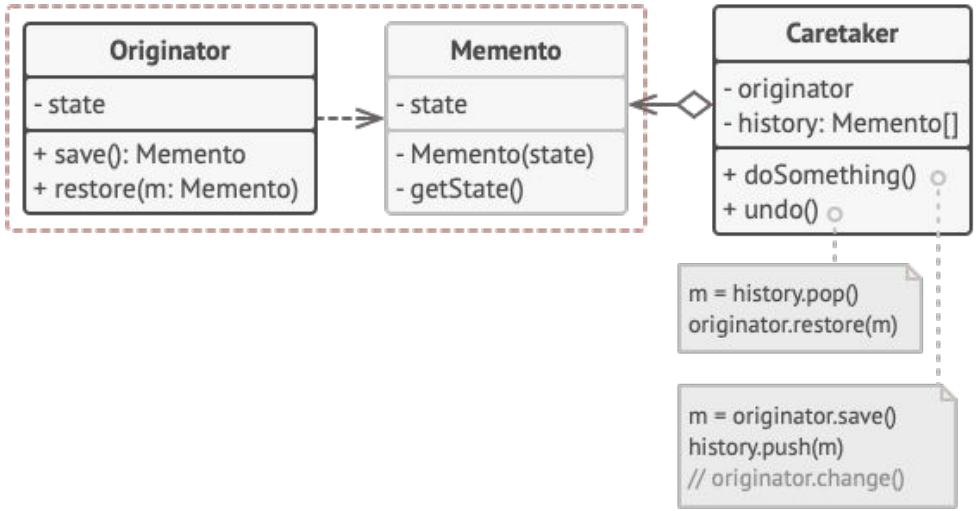
Implementácia



Implementácia



Využitie vzoru



Výhody a nevýhody

+ Rieši problém zapúzdrenia

+ Caretaker uchováva históriu stavu Originator-a, vďaka čomu je kód Originator-a jednoduchší

- Môže spotrebovať veľa pamäte RAM

- Väčšina dynamických programovacích jazykov (PHP, Python a JavaScript) nemôže zaručiť, že stav v Memente zostane nedotknutý

Zdroje

- https://refactoring.guru/design-patterns/behavioral-patterns
- https://refactoring.guru/design-patterns/mediator
- https://refactoring.guru/design-patterns/memento
- https://refactoring.guru/design-patterns/observer
- https://refactoring.guru/design-patterns/state

Ďakujeme za pozornosť

Otázka na skúšku

Vyberte správne tvrdenia ohľadom Mediator a Observer patternu:

✓ Ak komponent urobí nejakú zmenu, Observer upozorní všetky komponenty, ktoré sú na neho napojené.

X Ak komponent urobí nejakú zmenu, Mediator upozorní všetky komponenty, ktoré sú na neho napojené.

✓ Mediator rozhodne, ktorý komponent má reagovať na zmenu iného komponentu.

X Observer rozhodne, ktorý komponent má reagovať na zmenu iného komponentu.

✓ Mediator eliminuje priame prepojenia a závislosti medzi komponentami.

✓ Observer nastavuje dynamické jednosmerné spojenie medzi komponentami.