Návrhové vzory

SSPŠ
PVA4
2025 / 2026
Ing. Matěj Cajthaml

Návrhové vzory

- Ověřená řešení
- Konkrétních problému
- "Reinventing the wheel"

- Existuje sada řešení, které by měl znát každý

- Společný jazyk, komunikace
- Ověřené řešení
- Flexibilita

- Poznáním typů zjistíte, že se používají všude

OTÁZKA

Co je to špagetový kód? Proč ho nechceme?

Kategorie

- **Vytvářecí / Creational**
- řeší způsob vytváření objektů
- kdy se objekt vytvoří, jak, s čím
- singleton, factory

- **Strukturální / Structural**
- zabývá se strukturou objektů
- vztahy mezi entitami
- proxy, adapter

- **Chování / Behavioral**
- zaměřuje se na komunikaci mezi objekty
- rozdělení a zodpovědnosti
- observer, strategy

Singleton

- = Jedináček
- Zaručuje, že třída má pouze jednu instanci
- Globální a jednotný přístup k této instanci
- Připojení k databázi, logger, konfigurace
- Často označováno jako anti-pattern
- Často static vlastnost na třídě
- Skrývá závislosti
- Obtížnější testování
- Porušuje princip jedné zodpovědnosti (viz později)

Singleton

```
class DatabaseConnection {
    private static instance: DatabaseConnection;

    private constructor() {
        console.log("Navazuji spojení s databází...");
    }

    public static getInstance(): DatabaseConnection {
        if (!DatabaseConnection.instance) {
            DatabaseConnection.instance = new DatabaseConnection();
        }
        return DatabaseConnection.instance;
    }

    public executeQuery(sql: string): void {
```

Observer

- = pozorovatel
- Obrácení závislosti volání
- Něco se stane, tak na tom místě zavolám vše potřebné
- Definujeme závislost 1:N
- Když se zavolá událost, závislí jsou upozorněni
- UI eventy
- Reaktivní programování
- Události v aplikaci, API

Observer

```
type Listener<T> = (data: T) => void;

class TypedEventEmitter<TEvents extends Record<string, any>> {
    private listeners: { [K in keyof TEvents]?: Listener<TEvents[K]>[] } = {};

    public on<K extends keyof TEvents>(event: K, listener: Listener<TEvents[K]>): voi
        if (!this.listeners[event]) {
            this.listeners[event] = [];
        }
        this.listeners[event]!.push(listener);
    }

    public off<K extends keyof TEvents>(event: K, listener: Listener<TEvents[K]>): vo
        if (!this.listeners[event]) return;
        this.listeners[event] = this.listeners[event]!.filter(l => l !== listener);
```

Proxy

- = Zástupce
- Prostředník, který řídí přístup k původnímu objektu
- Provedení akce před přístupem na originál

- Virtual proxy: lazy loading
- Protection proxy: přístupová práva
- Remote proxy: zastupuje objekt na jiném serveru
- Logging proxy: loguje přístupy

Proxy

```
interface IDocumentService {
    getSecretDocument(id: number): string;
}

class RealDocumentService implements IDocumentService {
    getSecretDocument(id: number): string {
        console.log(`Stahuji tajný dokument č. ${id} z databáze...`);
        return "TOP SECRET DATA: Plány ke sklepu";
    }
}

class SecurityProxy implements IDocumentService {
    private realService: RealDocumentService;
    private userRole: string;

```

Další vzory

- Builder
- Prototype

- Adapter
- Bridge
- Facade
- Flyweight

- Iterator
- Memento
- Command
- Strategy
- Visitor

- https://refactoring.guru

Vlastní návrhové vzory

- Návrhové vzory nejsou "standard"
- Můžete si vytvořit vlastní
- Pokud ve vaší doméně řešíte něco speciálně

- Obecný problém over engineeringu

Refactoring

- Refactoring je změna struktury kódu
- Nemění funkcionalitu, na venek vypadá pořád stejně

- Nikdy není cíl dokonalý kód
- Cílíme na udržitelnost a čitelnost

- Kdy refaktorovat:
- code smells - duplicitní kód, dlouhé metody, nepřehledný kód
- před přidáním nové funkcionality
- při code review
- při opravě chyby

- Pravidlo 3

OTÁZKA

Zjistěte, jaké jsou další code smells.

OTÁZKA

Proč chceme čistý kód?

OTÁZKA

Co je to technologický dluh a jak vzniká?

Principy čistého kódu

- DRY - Dont repeat yourself
- KISS - keep it simple, stupid
- YAGNI - you ain't gonna need it

- SOLID
- Single responsibility: jedna zodpovědnost
- Open/Closed: otevřeno pro rozšíření, uzavřeno pro modifikaci
- Liskov substitution: podtřídy jsou zaměnitelné za rodiče
- Interface segregation: více rozhraní je lepší než jedno velké, nezávislost na všem
- Dependency inversion: záviset na abstrakcích => dependency injection

- Ukázka na webovém serveru, co vše potřebuje controller

Struktura projektu

- Kvalitní architektura => udržitelný projekt
- 3 vrstvé architektury, MVC, MVP, MVVM
- Rozdělení logiky, zobrazení a dat a proč

Psaní kódu

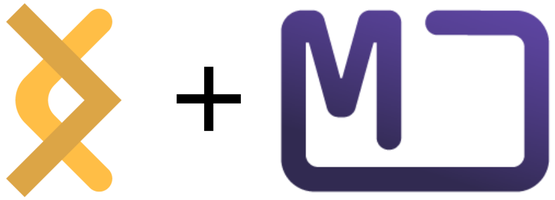
- Prvně napište kód, který je funkční a vše splňuje
- Refakturujeme až když je to složité, nebo něco opakujeme

PRÁCE

Otevřete svůj starý kód a najděte tam místa, kde
by se hodilo upravit kód a refaktorovat ho.

Děkuji za pozornost

matej.cajthaml@ssps.cz
ssps.cajthaml.eu



Vytvořeno pomocí aplikace Materialist