**Návrhové vzory - creational design patterns, structural design patterns, behavioral patterns**

Typické/obecné řešení pro časté a známé problémy při vývoji SW.

Koncept/šablona který se musí předělat pro vlastní použití a řešení problému.

OOP návrhové vzory typicky ukazují vztahy mezi třídy a objekty.

Nejsou jen z programování, jsou také v každodenním životě

* Např. architektura (stavby)

**!! ALGORITMY – Nejsou návrhové vzory, ty řeší konkrétní problémy !!**

**ZÁKLADNÍ TYPY**

* **Creational patterns (vytvářející)**
  + Vytváření objektů v systému
    - Znovu použitelnost
  + Dynamická rozhodnutí za běhu programu
  + Příklady
    - Singleton, Factory method, Abstract Factory
* **Structural patterns (strukturální)**
  + Uspořádání jednotlivých tříd nebo komponentů
  + Důvody
    - Zpřehlednit systém
    - Strukturalizace kódu
  + Příklady
    - Adapter, Decorator, Proxy, Bridge
* **Behavioral patterns (chování)**
  + Chování mezi objekty
  + Povinnosti objektů
  + Objekty
    - Spolupráce mezi objekty
    - Spolupráce mezi skupiny objektů
  + Příklady
    - Observer, Command, Iterator, State

**CREATIONAL PATTERNS**

**Factory method**

**Záměr**

Metoda factory je návrhový vzor, který poskytuje rozhraní pro vytváření objektů v nadřazené třídě, ale umožňuje podtřídám měnit typ objektů, které budou vytvořeny.

Podtřídy sami rozhodují, jaký typ objektu vytvoří.

**Příklad**

Vzor metody Factory navrhuje nahradit volání přímé konstrukce s voláním speciální tovární metody. Objekty jsou stále vytvářeny prostřednictvím new operátora, ale ten je volán z tovární metody.

**Struktura**

**Diagram

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidenceKód**

**Abstract factory**

**Záměr**

Abstract factory je návrhový vzor, který umožňuje vytvářet objekty bez určení jejich konkrétních tříd a mít u toho jistotu, že budou související.

**Problém**

Nábytek. Více typu nábytku. Při objednání židlí se objednává i pohovka. Pohovka by správně měla být stejného typu (stylu) jako židle.

**Řešení**

Vytvořit interface s metodami na vytvoření nábytku. Oddědit na třídy různých typů nábytku.

Když se volá tvorba židlí z jedné továrny, bude se tvořit nábytek jedného typu.

**Struktura**

****Diagram

Description automatically generatedKod

**Singleton**

**Záměr**

Singleton je návrhový vzor, který umožňuje zajistit, aby třída měla pouze jednu instanci, a zároveň poskytuje globální přístupový bod k této instanci.

**Problém**

**Singletonův vzor řeší dva problémy současně:**

Třída má pouze jednu instanci.

* řízení přístupu k nějakému sdílenému prostředku, například k databázi nebo souboru

Poskytněte globální přístupový bod k této instanci.

* Při globálních parametrech (bez použití singletonu) jakýkoli kód může potenciálně přepsat obsah těchto proměnných a havarovat aplikaci

**Řešení**

U singletone

* nastavte výchozí konstruktor jako soukromý, abyste zabránili ostatním objektům v použití operátoru new s třídou Singleton
* vytvořte statickou metodu vytvoření, která funguje jako konstruktor.
  + Pod kapotou tato metoda volá soukromý konstruktor k vytvoření objektu a uloží jej do statického pole.
* Všechna následující volání této metody vrátí objekt uložený v mezipaměti.

Struktura

**Diagram

Description automatically generated**

**Structural patterns**

**Decorator**

**Záměr**

Decorator je konstrukční návrhový vzor, který umožňuje přidat nové funkce objektům tím, že je umístí do speciálního “wrapperu” (obalu).

**Problém**

Vytváříte notifikace. Tato metoda má message argument od klienta, který pošle e-mailům v listu, které byly uloženy přes konstruktor.

Při jiných typech oznámení potřebujete posílat také jiné oznámení.

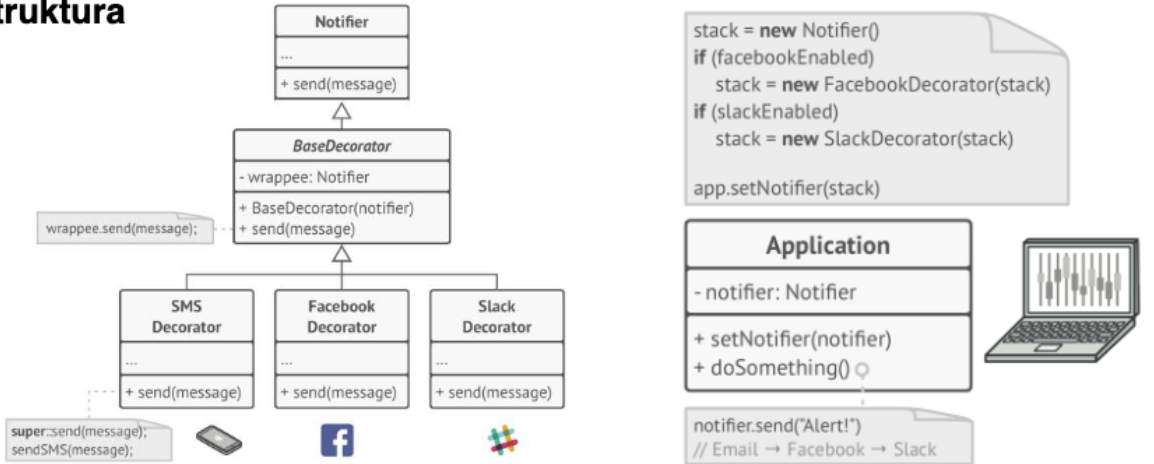
**Řešení**

Zaměňuje dědění.

Decorator/Wrapper

* objekt, který je spojený s cílovým objektem
* obsahuje stejné metody jako cíl a všechny příchozí volání přeposílá cíli
* ještě před odesláním je schopný příchozí žádost upravit.

**Struktura**

**Kod**

**Proxy**

**Záměr**

Proxy

* meziobjekt, který je mezi cílovém objektu a všech ostatních
  + meziobjekt ovládá přístup k cílovému objektu
  + umožňuje provést libovolné změny buď před nebo poté co se dotaz dostane skrz cílový objekt.

**Problém**

Do objektu chceme přidat další funkce, ale může být v uzavřený knihovně. Lze přidat do každé třídy = duplicitní kód a práce navíc.

**Řešení**

Vytvoří nový proxy objekt

* stejný interface jako cílový objekt
* aktualizovat aplikaci, aby používala nový objekt místo napřímo starý
* do metody objektu vepsat podmínky/změny,

**Struktura**

**A picture containing text

Description automatically generated**

**Kód**

**Graphical user interface, text

Description automatically generated**

**Behavioral patterns**

**Command**

**Záměr**

Command

* mění request do samostatného objektu
  + objekt obsahuje všechny informace o requestu
* toto umožňuje
  + přidat parametry metodám s různými requesty
  + zpoždění splnění requestu
  + vložení requestů do fronty.

**Problém**

Třída BUTTON, ale víc tlačítek s jinou funkcionalitou. Kam ale dáme logiku pro jednotlivá tlačítka, když každé z nich má jinou funkci? Nejjednodušší je zdědit třídu Button pro každé tlačítko s různou logikou. Toto ale není nejlepší řešení, protože kvůli tomu vytvoříme obrovské množství tříd a existuje tu risk při změně originální Button třídy.

**Řešení**

Rozdělit aplikaci na vrstvy. V tomto případě ji rozdělit na GUI vrstvu a vrstvu na logiku aplikace. V kódu by to vypadalo tak, že GUI objekt zavolá metodu z logické vrstvy s argumenty. Tomuto procesu se říka poslání requestu. Command v tomto případě je mezivrstvou mezi GUI a logikou tím, že GUI spustí command a ten se postará o všechny detaily. Ten by měl mít již přístup ke všem datům requestu a nebo být schopný si tato data obstarat.

**Diagram

Description automatically generatedStruktura**

**Ukázka kódu**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Observer**

**Záměr**

Vzor návrhu chování, který umožňuje definovat funkci předplatného = upozornění více objektů na události, ke kterým dojde u objektu, který pozorují.

**Problém**

Dva typy objektů: Customer a Store.

Zákazník má velký zájem o zboží, který by mělo být k dispozici v obchodě velmi brzy. Ale posílat oznámení o zboží každému nedává smysl.

**Řešení**

do třídy vydavatele přidat mechanismus předplatného

* jednotlivé objekty mohou přihlásit k odběru nebo se odhlásit z odběru událostí
* Tento mechanismus se skládá z
  + 1) pole pro uložení seznamu odkazů na objekty předplatitelů
  + 2) veřejný metody pro přidání a odebrání předplatitele
* kdykoli se vydavateli stane odebíraná událost
  + zavolá konkrétní metodu oznámení na objekty předplatitelů
  + !! aby všichni předplatitelé implementovali stejné rozhraní
  + !! aby s nimi vydavatel komunikoval pouze prostřednictvím tohoto rozhraní.

**Struktura**

**Diagram

Description automatically generated**

**Ukázka kódu**

**Graphical user interface, text, application, chat or text message

Description automatically generated**