

# OBJEKTIVĚ ORIENTOVANÉ PROGRAMOVÁNÍ A POUŽITÍ OBECNÝCH NÁVRHOVÝCH VZORŮ

(ODBORNÝ VÝKLAD)

MARTIN HUJER

OSNOVA:

- I. Výhody objektově orientovaného programování
- II.
  1. Základní prvky OOP
    - a) třída
    - b) objekt
    - c) metoda
    - d) vlastnost
  2. Základní principy OOP
    - a) zapouzdření
    - b) dědičnost
    - c) polymorfismus
  3. Návrhové vzory
    - a) účel a princip použití návrhových vzorů
    - b) konkrétní příklady návrhových vzorů
      - i. návrhový vzor Singleton
      - ii. návrhový vzor Factory
- III. Shrnutí výhod OOP a využití návrhových vzorů

Objektově orientované programování (běžně se pro něj využívá zkratka OOP) je moderní způsob zápisu programového kódu, který nahrazuje dříve všeobecně používané tzv. procedurální programování. OOP umožňuje snadnější strukturování kódu a jeho rozčleňování do logicky souvisejících bloků.

OOP využívá tyto prvky: třída, objekt, metoda, vlastnost.

Třída definuje charakteristiky předmětu (objektu) včetně jeho vlastností (property) a principů chování (co třída může provádět, metody, operace). Je možno říci, že třída je návrh, který definuje charakter objektu. Třídy poskytují strukturu a modularitu při tvorbě programového kódu především zapouzdřením (*encapsulation*; viz. níže).

Objekt lze definovat jako instanci konkrétní třídy. Jestliže třídu považujeme za návrh, jak má objekt vypadat, tak objekt je entita vytvořená podle návrhu (třídy). Například existuje postup pro výrobu stolu (v našem případě třída) a podle něj vytvoříme konkrétní stůl (naš objekt). Zkráceně: Objekt je instancí třídy.

Metodou se označuje jakákoliv akce nebo operace, kterou objekt může provádět. Metoda může manipulovat s vnitřními vlastnostmi objektu a také může volat veřejné metody jiných objektů. U stolu se dají očekávat metody jako *OtevřiŠuplík*, *ProstřiSe* a podobně.

Vlastnost popisuje konkrétní parametry daného objektu. Například u našeho stolu můžeme definovat jako parametry šířku a délku desky, výšku noh, barvu, materiál a další.

Základní prvky OOP se vzájemně propojují a tvoří základní principy OOP: zapouzdření, dědičnost a polymorfismus.

Zapouzdření lze vysvětlit například tak, že konkrétní objekt poskytuje navenek nějaké metody, které manipulují s vlastnostmi objektu. Vlastnostem nebo metodám lze nastavit viditelnost, standardně `public`, `protected` a `private`. `Public` znamená, že metoda nebo vlastnost je viditelná a použitelná zvenku třídy. K `protected` vlastnosti nebo metodě je možné přistupovat jen ve zděděných třídách. Vlastnosti nebo metody označené `private` jsou přístupné pouze v dané třídě. Takže například u našeho stolku nenastavujeme přímo vlastnost `material`, ale použijeme metodu `setMaterial(barva)`. Slovo `set` je prakticky standard a používá se i u projektů, které mají vlastnosti a metody pojmenované česky.

Dědičnost umožňuje vytvořit nejprve obecnější třídu a od ní poté odvodit další třídy, které přebírají vlastnosti a metody rodičovské třídy. Například výše `Stůl`, by byl pravděpodobně vytvořen jako potomek třídy `Nábytek`, která by měla vlastnosti jako `materiál` nebo `výška`, `šířka` a `délka`. `Stůl` by navíc přidával vlastnosti jako `početNohou` nebo `mášuplík` a například metodu `OtevřiŠuplík`. Jedním z principů dědičnosti je `overloading` (překrytí) vlastností nebo metod v odvozené třídě.

Polymorfismus lze popsat tak, že objekty různých typů odpovídají na volání shodně pojmenovaných metod rozdílnými způsoby závislémi na typu konkrétního objektu. Způsob implementace tohoto chování se v jednotlivých programovacích jazycích liší. Některé jazyky umožňují využití `Interface` (rozhraní), které obsahuje definice abstraktních metod. Toto rozhraní potom jednotlivé objekty implementují. V jiných jazycích je nutno vytvořit abstraktní třídu, od které jsou poté odvozeny jednotlivé třídy. Pojem abstraktní znamená, že od třídy nelze přímo vytvořit instanci, ale je nutno nejdříve tuto třídu zdědit, překrýt všechny definované metody a poté vytvořit instanci této zděděné třídy. Abstraktní metody nemohou být volány přímo, ale je také nutné je v odvozené třídě překrýt.

Návrhový vzor představuje obecné řešení problému, který se opakovaně objevuje při návrhu softwaru. Návrhový vzor není knihovnou nebo částí zdrojového kódu, kterou by bylo možné přímo využít v naší aplikaci. Jedná se o popis či šablonu, jak řešit problém způsobem, který může být použit v různých situacích. Objektově orientované návrhové vzory zobrazují vztahy a interakce mezi třídami a objekty, aniž by určovaly implementaci konkrétní třídy.

Návrhové vzory nepocházejí ze softwarového inženýrství, ale jsou naprosto běžné v každodenním životě. Typickým příkladem je například architektura. Gotickou katedrálu poznáte právě proto, že tehdejší architekti využívali návrhové vzory.

Použití návrhových vzorů zdokonalilo na počátku devadesátých let minulého století uskupení nazývané `Gang of Four` (běžně se využívá zkratka `GoF`), které tvořili Erich Gamma, Richard Helm, Ralph Johnson a John Vlissides.

Návrhových vzorů existuje velké množství, ale některé se používají mnohem častěji než ostatní.

Návrhový vzor `Singleton` omezuje možnost vytváření instance dané třídy. Při použití tohoto návrhového vzoru existuje vždy jen jediná instance dané třídy. Tohoto se využívá u objektů, které mohou existovat jen jedenkrát, například jedno spojení s databází nebo například zápis do souboru, který může být otevřen jen jedenkrát. `Singleton` má konstruktor označený jako `private`, tudíž ho nelze zavolat přímo. Místo toho `singleton` disponuje statickou metodou `getInstance()`, která kontroluje existenci instance daného objektu. Pokud neexistuje, tak ji

vytvoří, uloží si ji do vlastní proměnná a vrátí ji. Tím je zajištěno, že vždy bude existovat jen jedna instance.

Návrhový vzor Factory patří mezi tzv. creational patterns a tudíž se zabývá vytvářením objektů. Umožňuje rozhodnout až v průběhu běhu aplikace o třídě, jejíž instance se vytvoří. Například mějme aplikaci, která umožňuje připojení k různým RDBMS (Relational database management systém - Systém řízení báze dat). Máme třídy `DatabaseMysql` a `DatabaseOracle`, které obě implementují rozhraní `Database` a tudíž je možné je využít pro práci s databázovým systémem. O tom, která z těchto tříd se použije, rozhoduje hodnota nastavená například v konfiguračním souboru. Volání faktory třídy probíhá většinou tímto způsobem: `DatabaseFactory::factory('mysql')`. Třída `DatabaseFactory` rozhodne na základě parametru předanému metodě `factory()`, jaká třída se má pro připojení využít.

Objektové programování usnadňuje návrh aplikací s ohledem na přehlednost programového kódu a reuseability (znovupoužitelnost) jednotlivých komponent. Znovupoužitelnosti lze dosáhnout především rozdělením tříd do logických bloků a zapouzdřením. Při použití OOP lze snadno kreslit diagramy závislostí tříd (pomocí UML - Unified Modeling Language).

Návrhové vzory nám radí, jakým osvědčeným způsobem je nejhodnější řešit daný problém, není potřeba vše vymýšlet znovu od naprostých základů.

#### **Zdroje:**

- Vlastní zkušenosti a praxe
- [http://en.wikipedia.org/wiki/Object-oriented\\_programming](http://en.wikipedia.org/wiki/Object-oriented_programming)
- Design Patterns, Elements of Reusable Object Oriented Software (Gang of Four - Erich Gamma, Richard Helm, Ralph Johnson a John Vlissides)
- [http://en.wikipedia.org/wiki/Design\\_pattern\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science))
- PHP 5 Power Programming (Andi Gutmans, Stig Sæther Bakken a Derick Rethans)
- php|architect's Guide to PHP Design Patterns (Jason E. Sweat)