# 1. Úvod do objektově orientovaného programování

Dostala se Vám do ruky publikace pojednávající o objektově orientovaném programování v jazyku Java. Předpokladem pro čtení je základní znalost syntaxe jazyka a zároveň dovednost práce se základními řídícími strukturami programování – větvením a cykly.

Oproti strukturovanému programování známému např. z programovacích jazyků C či Pascal se v rámci OOP snažíme o úplně jiný náhled na program jako takový. Budování programu se mnohem více podobá skutečné realitě.

Vysvětleme to na příkladu: Vlastníme automobil. Určitě je skvělé, že si nemusíme náhradní díly vyrábět sami. Že lze zajet do pneuservisu a pořídit si hotovou a otestovanou pneumatiku. Můžeme koupit správné brzdové destičky, či nalézt a koupit správně padnoucí světlomet. Lze říci, že tyto „objekty“ jsou již někde hotovy a my je nemusíme vyrábět sami. Jen si je pořídíme a můžeme je využívat bez obav, že by bylo něco v nepořádku.

Stejně tak programátoři na celém světě programovali a programují různorodé komponenty, které pak ostatní vývojáři mohou využívat. Zdarma či za úplatu. Dokonce se tvrdí, že téměř vše důležité již bylo někdy v minulosti naprogramováno, a tedy hlavní um tkví v tom klíčové entity vyhledat, naučit se je používat a poskládat je do požadovaného funkčního celku.

To by ale nebyla taková zábava, a proto v této učebnici budeme především programovat od základů vlastní objekty. A to tak, abychom pochopili hlavní struktury a zákonitosti OOP. Tyto pak budeme moci aplikovat i v ostatních objektově orientovaných jazycích, nejen striktně v Javě.

OOP přístup nám zjednodušuje pohled na složitější a delší programy. V případě strukturovaného programování může mít program tisíce řádků v jediném souboru. Stává se tak nepřehledným a těžko editovatelným. Oproti tomu objektový přístup pracuje s jednotlivými funkčními celky. Programovaný kód je rozčleněn a stává se tak mnohem čitelnějším. Lze se v něm jednoduše vyznat i v případě tisíců řádků kódu.

## Příklad

Pojďme si uvést další výhodu objektového přístupu na praktické ukázce. Máme za úkol do programu implementovat informace o různých zaměstnancích. Pokud bychom programovali strukturovaně, zřejmě bychom vytvořili několik polí. První pole by sloužilo k ukládání jmen, další pole k ukládání opracovaných let, další pak například k uložení hrubé hodinové mzdy zaměstnance... Informace o jednom daném zaměstnanci by tedy neměly v paměti žádnou souvislost. Spojoval by je pouze index – číslo, pod nímž by byly související informace uloženy v jednotlivých polích. Např. pod indexem 3 by byl uložen Petr Novák, který by měl v dalším poli na indexu 3 uloženo, že pracuje ve firmě již 12 let. V dalším poli týkajícího se mezd by bylo na indexu 3 uloženo, že zaměstnanec pobírám mzdu 200 Kč hrubého za hodinu.

Z řešení plyne jedna zásadní nevýhoda – pokud by teoreticky v jednom jediném poli došlo k posunu hodnot a v ostatních nikoli, veškeré informace o zaměstnancích by byly nesprávné – dočista promíchané,  
 a tedy nepoužitelné.

## Zapouzdření

Nyní řešme výše uvedený příklad pomocí OOP. Naši práci zahájíme vytvořením tzv. **třídy. Jedná se o** strukturovaný celek pro uložení zaměstnance jako takového. Jako jeho součást definujeme proměnné pro uložení jména, odpracovaných let i mzdy. Informace jsou tedy uloženy pospolu pod jediným objektem – objektem třídy Zamestanec.java. V tomto případě nám tedy stačí pro potřeby ukládání vytvořit pouze jediné pole – pole pro ukládání objektů typu Zamestnanec. Správné informace se budou držet pospolu v rámci jediného objektu a nemůže tedy dojít k promíchání, jako tomu bylo ve struktuře.

Přidejte si k tomu výhodu, že můžete mít plnou kontrolu nad tím, které vlastnosti zaměstnance budou viditelné a které editovatelné. Pokud existuje mezi položkami v rámci jednoho objektu souvislost, můžete ulehčit uživateli práci a některé funkce zautomatizovat. Kupříkladu hrubá mzda závisí na počtu odpracovaných let. Pokud programovanou třídu upravíte dle tohoto požadavku, hrubá mzda se bude měnit automaticky a uživatel tedy nemůže nepozornostní způsobit chybu.

Gratuluji, právě jste pochopili pojem **zapouzdření objektu**.

## Dědičnost

Nyní si představte, že Váš klient přišel s novým požadavkem. Zaměstnanců je více typů. Např.  
 ve školství zaměstnanci pedagogičtí a nepedagogičtí. Tyto dvě entity se liší minimálně, ale rozdíly zde jsou. Abyste nemuseli objekty znovu programovat od základů či kopírovat části kódu, nabízí OOP další výhodu – **třídy lze tvořit na základě jiných, již hotových tříd**. V našem případě nejprve definujeme zaměstnance  
 s atributy, které jsou pro všechny typy zaměstnanců stejné. Např. jméno, rodné číslo či hrubá mzda. Následně na základě této třídy definujete zaměstnance pedagogického. Ten zdědí veškeré atributy, ale bude zde přidán atribut aprobace. Do třídy nepedagogického zaměstnance doprogramujeme navíc pracovní zařazení. Ostatní atributy se rovněž použijí (zdědí) z původní třídy zaměstnanec.

Pojďme poukázat na jednu ne zcela zřetelnou výhodu. Předpokládejme, že bychom chtěli změnit strategii přidělování peněz dle odpracovaných let. Stačilo by tuto funkcionalitu upravit ve třídě zaměstnanec. Třídy pedagogických a nepedagogických zaměstnanců by pak toto chování automaticky převzaly, zdědily by jej od předka. Pro programátora to znamená méně úprav, méně možných chyb a přehlédnutí, méně duplicitního kódu.

Došli jste tedy k dalšímu důležitému pojmu OOP – **dědičnost**.

## Polymorfismus

Předpokládejme nyní, že dědičnost v našem programu funguje. Jako programátoři chceme, aby všechny třídy v této dědické struktuře vykonávaly stejné nebo podobné funkce a také aby se tyto funkce (lépe řečeno **metody**) stejně jmenovaly. Např. VratInfo, nebo UpravPocetOdpracovanychLet. Toto nám dědičnost umožní. Nebudeme tak naše metody muset pojmenovávat např. VratInfoZamestnanec, VratInfoPedagogicky, VratInfoNepedagogicky. Všude bude pouze VratInfo, program bude určitě lépe čitelný i použitelný. Tedy máme 3 metody, které se shodně jmenují, ale každá je v jiném objektu dědické struktury. V těle každé z těchto 3 metod můžeme naprogramovat libovolný kód ušitý na míru danému objektu.

Nyní jsme osvětlili pojem **polymorfismus**. Slovo zní složitě, ale není to nic jednoduššího než programování stejně nazvaných metod s různou funkcionalitou v objektech spolu dědicky souvisejících.

Seznámili jsme se tedy se 3 základními vlastnostmi programování objektově orientovaných programů:

* Zapouzdřenost
* Dědičnost
* Polymorfismus

Všechny vlastnosti OOP budeme mít během naší práce v povědomí a budeme s nimi počítat. Nyní můžeme směle pokračovat k další kapitole, kde budeme konečně i programovat.