# 3. OOP – Třídy, vlastnosti a metody

# OOP

Objektově orientované programování (OOP) je [programovací paradigma](http://cs.wikipedia.org/wiki/Programovací_paradigma). Rozdíl oproti procedurálnímu programování nespočívá jen v psaní kódu jiným způsobem, ale v uvažování jiným způsobem. Cílem OOP je vytvářet malé, relativně samostatné a v různých aplikacích opakovaně použitelné jednotky (což, jak uvidíme níže, jsou právě „objekty“).

Jedná se o filozofii a způsob myšlení, designu a implementace, kde klademe důraz na **znovupoužitel­nost**.

# Proč programovat objektově

Jako výhody se uvádějí snazší a rychlejší vývoj aplikací, snazší údržba a menší chybovost. Nevýhodou je, že správa objektů zabere určité systémové prostředky, takže dokonale napsaný objektový program bude pomalejší, než dokonale napsaný procedurální program. Zastánci objektového programování ale argumentují, že prakticky žádná reálná aplikace nebude dokonale napsaná, přičemž v OOP je díky jeho vlastnostem snazší se té dokonalosti přiblížit, čímž se eliminuje ta výkonová nevýhoda.

V každém případě ale v dnešní době u mnoha aplikací roste důležitost snadné údržby a rozšiřitelnosti na úkor výkonové optimalizace. Zatímco výpočetní výkon často lze jednoduše (třeba i jen posunutím šoupátka v administraci) a levně (ve srovnání s cenou práce programátora) navýšit posílením hardwaru, neschopnost opravovat chyby aplikace a přidávat nové vlastnosti může být fatální.

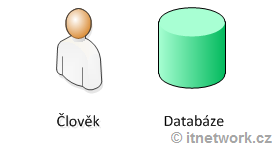
# Jak OOP Funguje

Snažíme se nasimulovat realitu tak, jak ji jsme zvyklí vnímat. Můžeme tedy říci, že se **odpoutáváme od toho, jak program vidí počítač (stroj) a píšeme program spíše z pohledu programátora (člověka)**. Jako jsme tehdy nahradili assembler lidsky čitelnými matematickými zápisy, nyní jdeme ještě dál a nahradíme i ty. Jde tedy o určitou úroveň abstrakce nad programem. To má značné výhody už jen v tom, že je to pro nás přirozenější a přehlednější.

# Stavba OOP

## Objekt

Základní jednotkou je objekt, který odpovídá nějakému objektu z reálného světa (např. objekt člověk nebo databáze).



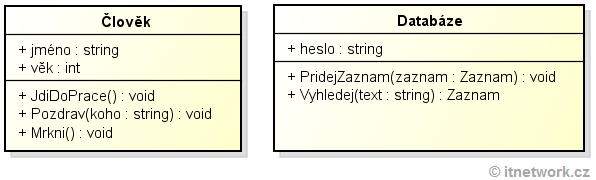
Objekt má své atributy a metody.

### Atributy

**Atributy objektu jsou vlastnosti** neboli data, která uchovává (např. u člověka jméno a věk, u databáze heslo). Jedná se o prosté proměnné, se kterými jsme již stokrát pracovali. Někdy o nich hovoříme jako o vnitřním stavu objektu.

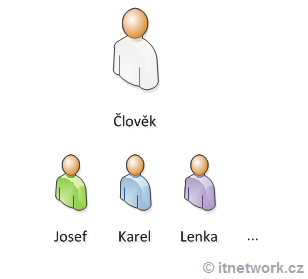
### Metody

**Metody jsou schopnostmi**, které umí objekt vykonávat. U člověka by to mohly být metody: JdiDoPrace(), Pozdrav() nebo Mrkni(). U databáze PridejZaznam() nebo Vyhledej()). Metody mohou mít parametry a mohou také vracet nějakou hodnotu. Velmi dobře je známe, používali jsme např. metodu Split() na objektu string. String je vlastně objekt, který reprezentuje v realitě nějaký text. Vidíte, že si můžeme jednoduše představit, že jednáme s řetězcem textu, něco mu přikazovat nebo na něm něco nastavovat. Obsahuje metody, které řetězec umí vykonávat (kopírování, mazání, splitování...) a má také nějaké atributy, např. Length, který značí jeho délku.



## Třída

Třída se dá chápat jako soubor příkazů. Třída však umožňuje mnohem více. Třída je vzor, podle kterého se objekty vytváří. Definuje jejich vlastnosti (Atributy) a schopnosti (Metody).

Objekt, který se vytvoří podle třídy, se nazývá instance. Instance mají stejné rozhraní jako třída, podle které se vytváří, ale navzájem se liší svými daty (atributy). Mějme například třídu Člověk a od ní si vytvořme instance karel a josef. Obě instance mají jistě ty samé metody a atributy, jako třída (např. jméno a věk) a metody (JdiDoPrace() a Pozdrav()), ale hodnoty v nich se liší (první instance má v atributu jméno hodnotu "Karel" a ve věk 22, druhá "Josef" a 45).

Komunikace mezi objekty probíhá pomocí předávání zpráv, díky čemuž je syntaxe přehledná. Zpráva obvykle vypadá takto: příjemce.Jméno­Metody(parame­try). Např. karel.Pozdrav(sousedka) by mohl způsobit, že instance karel pozdraví instanci sousedka.

OOP stojí na základních třech pilířích: Zapouzdření, Dědičnost a Polymorfismus.

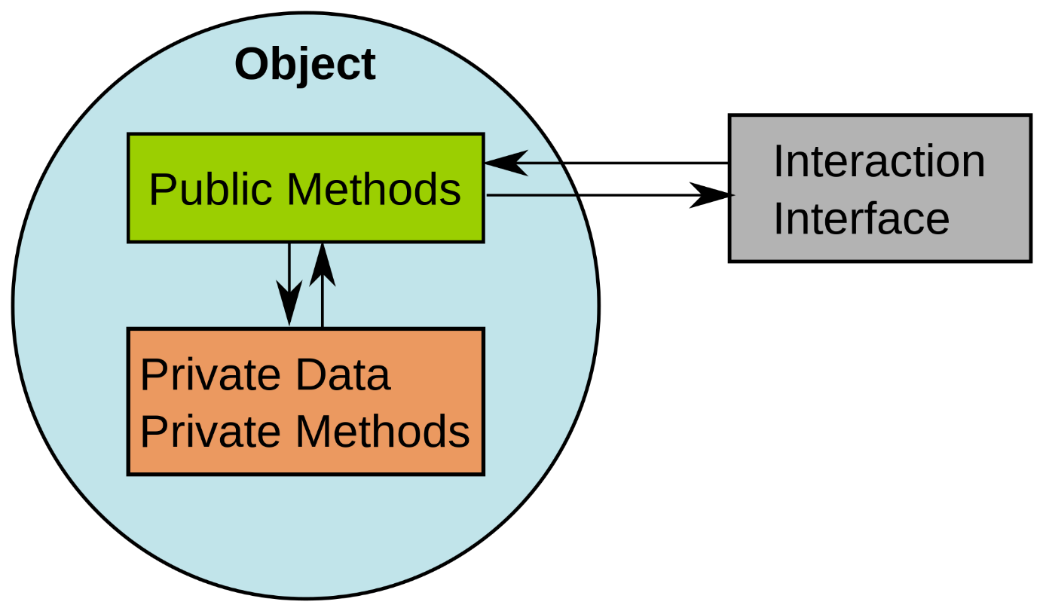
# Zapouzdření

Zapouzdření umožňuje **skrýt některé metody a atributy** tak, aby zůstaly použitelné jen pro třídu zevnitř. Objekt si můžeme představit jako **černou skřínku** (anglicky blackbox), která má určité **rozhraní (interface)**, přes které jí předáváme instrukce/data a ona je zpracovává.

Nevíme, jak to uvnitř funguje, ale víme, jak se navenek chová a používá. Nemůžeme tedy způsobit nějakou chybu, protože využíváme a vidíme jen to, co tvůrce třídy zpřístupnil.

Příkladem může být třída *člověk*, která bude mít atribut *datumNarozeni* a na jeho základě další atributy: *plnolety* a *vek*. Kdyby někdo objektu zvenčí změnil *datumNarozeni*, přestaly by platit proměnné *plnolety* a *vek*. Říkáme, že vnitřní stav objektu by byl nekonzistentní. Toto se nám ve strukturovaném programování může klidně stát. V OOP však objekt zapouzdříme a atribut *datumNarozeni* označíme jako privátní, zvenčí tedy nebude viditelný. Naopak ven vystavíme metodu ZmenDatumNaro­zeni(), která dosadí nové datum narození do proměnné *datumNarozeni* a zároveň provede potřebný přepočet věku a přehodnocení plnoletosti. Použití objektu je bezpečné a aplikace stabilní.

Zapouzdření tedy donutí programátory používat objekt jen tím správným způsobem. **Rozhraní (interface) třídy rozdělí na veřejně přístupné (public) a vnitřní strukturu (private)**.

[](https://www.google.cz/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwj5rMCxt_zLAhVL7xQKHUHvA30QjRwIBw&url=https://commons.wikimedia.org/wiki/File:CPT-OOP-interfaces.svg&psig=AFQjCNHxFOG43hBlJdGqDGlVa6gMqJNxvA&ust=1460115237436783)

# Dědičnost

Objekty mohou dědit své vlastnosti, tzn. zejména strukturu lokálních dat a množinu svých metod, od jim nadřazených objektů. Vzniká tak hierarchie objektů, přičemž nejobecnější objekty jsou v nižších stupních hierarchie postupně upřesňovány (a tím také rozmělňovány) do mnoha konkrétnějších tříd.

K popisu vztahů v hierarchii se používá pojmenování předchůdce-následník nebo rodič-syn.

Při programování se nemusí znovu opakovat části, které jsou zděděné; následník automaticky přebírá všechna data i metody svého předchůdce. (Polymorfismem je ovšem může změnit.)

|  |  |  |
| --- | --- | --- |
| **Příklad:** | **třída: OBRAZEC**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Atributy:** X,Y souřadnice   středu  Barva  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Metody:** přebarvit  vykreslit  smazat  přesunout |  |
|  |  |  |
| **Instance: OBDÉLNÍK**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Atributy:** šířka  výška  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Metody:** Zvětšit  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  | **Instance: KRUH**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Atributy:** Poloměr  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Metody:** Zvětšit  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

Zděděné třídy **OBDÉLNÍK** i **KRUH** obsahují všechna data, která obsahoval jejich rodič i všechny metody, které obsahoval jejich rodič. Navíc každá ze zděděných tříd obsahuje ještě i svá specifická data a metody. Zděděné třídy mohou obsahovat data a metody navíc, ale nemohou nic z rodičovské třídy ztratit!

Obě zděděné třídy mají své metody **Zvětšit**. Je třeba vidět, že Zvětšit třídy OBDÉLNÍK je jiná metoda než Zvětšit třídy KRUH. S každou třídou se volá její správná Zvětšit a nijak si nepřekážejí, i když se jmenují stejně.

Kdyby se stalo, že dceřiná třída má metodu se stejným jménem jako rodičovská třída, pak se v zásadě použije pravidlo, že "platí to lokálnější". To znamená, že bude platit metoda dceřiné třídy a metoda stejného jména patřící rodiči bude překryta, bude neviditelná. Jenže tento mechanismus může mít četné a velmi zajímavé modifikace, které jsou podstatou další vlastnosti OOP, která se nazývá [polymorfismus](http://www1.fs.cvut.cz/cz/u12110/oop/PREDNESY/04/polymorf.htm).

# Polymorfismus

Polymorfismus umožňuje používat jednotné rozhraní pro práci s různými typy objektů. Mějme například mnoho objektů, které reprezentují nějaké geometrické útvary (kruh, čtverec, trojúhelník). Bylo by jistě přínosné a přehledné, kdybychom s nimi mohli komunikovat jednotně, ačkoli se liší. Můžeme zavést třídu Obrazec, která by obsahovala atribut barva a metodu vykresli. Všechny geometrické tvary by potom dědily z této třídy její interface (rozhraní). Objekty kruh a čtverec se ale jistě vykreslují jinak. Polymorfismus nám umožňuje přepsat si metodu vykresli u každé podtřídy tak, aby dělala, co chceme. Rozhraní tak zůstane zachováno a my nebudeme muset přemýšlet, jak se to u onoho objektu volá.

Polymorfismus bývá často vysvětlován na obrázku se zvířaty, která mají všechna v rozhraní metodu Speak(), ale každé si ji vykonává po svém.



Podstatou polymorfismu je tedy metoda nebo metody, které mají všichni potomci definované se stejnou hlavičkou, ale jiným tělem.

# Kompozice

Podobná dědičnosti.

Máme třídy a objekty: Osoba, Pavel a Osoba, Datum narození.

Dědičnost: Je? – Je Pavel Osoba? Ano = Dědičnost

Kompozice: Má? – Má Osoba Datum narození? Ano = Kompozice