**Principy objektového programování, agregace a kompozice objektů**

**OOP (object oriented programming)**

Styl programování, kde je **postup, analýza návrh a implementace založena na přímém modelování objektů z reálného světa** (včetně vazeb a interakcí) ve světě počítače s využitím prostředků pro **abstrakci a hierarchizací**

Co umožňuje OOP programátorovi?

1. Abstrakci (zobecnění) pro modelování a řešení problémů
2. Znovupoužitelný kód.
3. Kontrola přístupu k datům.
4. Minimalizace samovolných / nezamýšlených zásahů do kódu.
5. Udržení pořádku v identifikátorech

* Řešení problému s využitím = analogem k lidskému postupu řešení
* Model objektu z reálného světa = objekt

**OBJEKT**

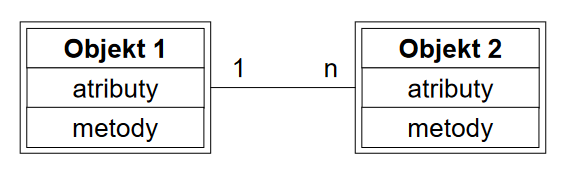
* datové struktury (atributy/proměnné)
* operace na objekt = metody
* vlastní život objektu = thread
* vzájemná komunikace = voláním metod
* Životnost objektu
  + Časový interval, ve kterém může být použit
  + Objekt je chápán jako lokální proměnná (alokován pamětí)
  + Nepotřebná paměť není uvolňována automaticky
    - **Garbage collector**
      * Automatické rušení objektů, na které neexistuje žádný odkaz

**TŘÍDA**

* Abstrakce chování a vlastností objektů
* Určuje, jak bude objekt vypadat a jak se bude chovat

**Objektově orientovaná analýza**

* hledání objektů v reálném světě
* hledání vazeb mezi objekty
* při analýze pouze data, nikoliv algoritmy
* výstupem analýzy = popis pomocí diagramů, např. UML



**Koncepty OOP**

* **Abstrakce**
* **Dědičnost**
* **Kompozice**
* **Polymorfismus**
* **Zapouzdření**

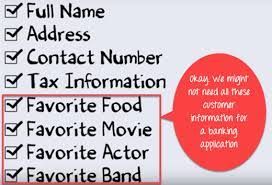
**Abstrakce**

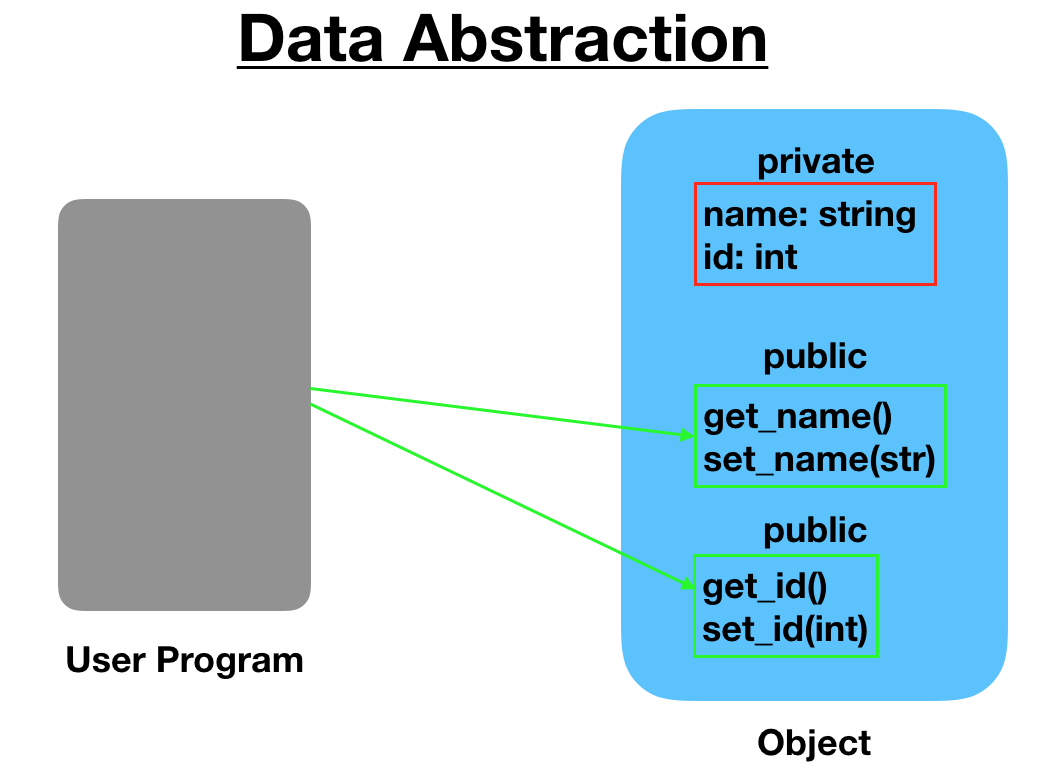
Proces **zjednodušení** složitých systémů **pomocí rozdělení na** více menších a lépe zvládnutelných **částí**

* Tvoření abstraktních tříd a metod

Zjednodušení zanedbáním méně podstatných detailů/vlastností

* Programátora nemusí zajímat detaily a jak vnitřně objekty pracují
  + Např. Jak pracuje knihovna, u který potřebujeme pouze její původní funkcionalitu
  + Pouze sdělení, co se má udělat, a ne jak se to má udělat



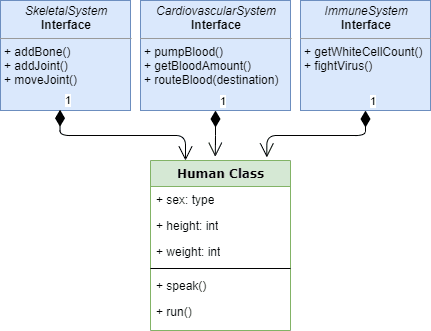


**Kompozice**

Výsledkem je **znovupoužitelný kód**

Složité objekty jako kolekce jednoduchých a akce redukovány na jednodušší

* **Postup zjednodušení, dokud objekty nejsou co nejvíc snadno použitelné a řešitelné**

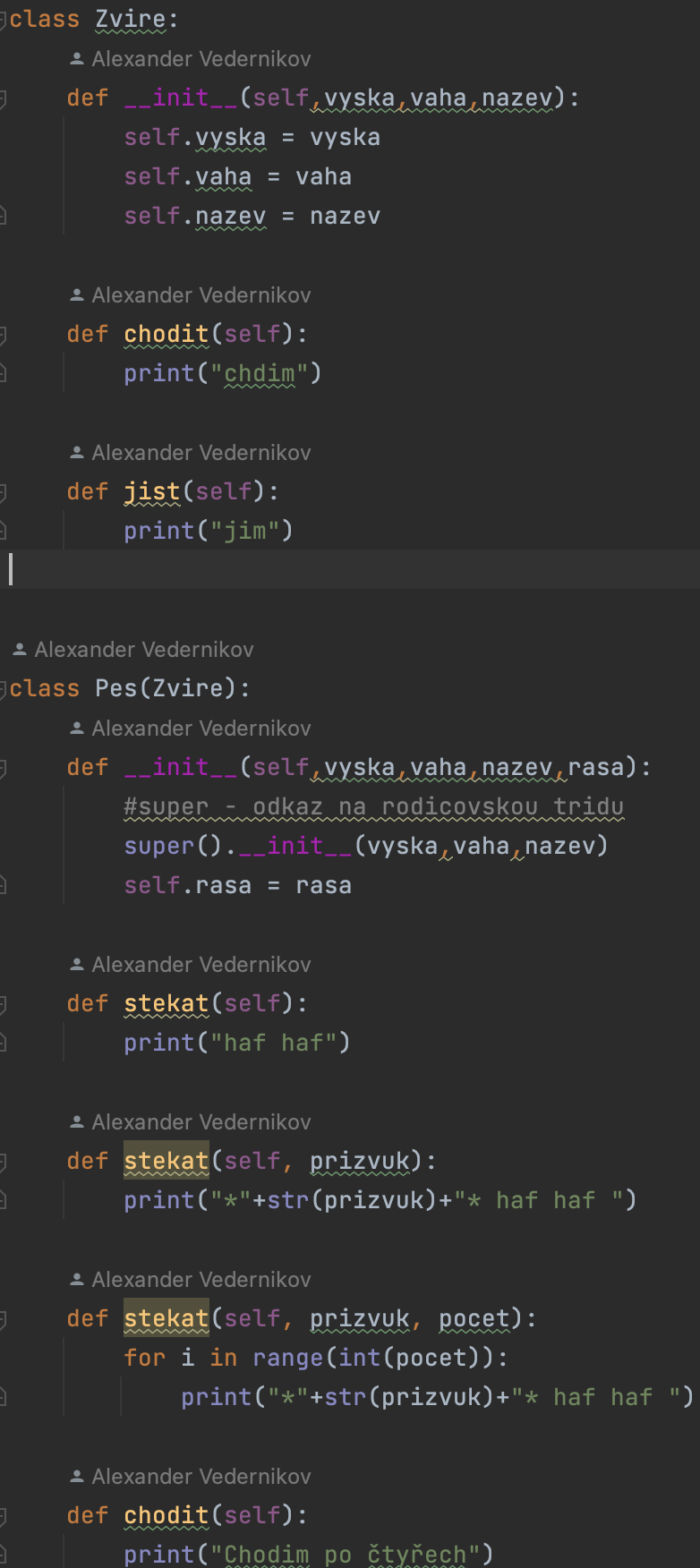


**Polymorfismus (mnoho tvarovost)**

Objekty reagují jinak na stejný podnět v závislosti na jejich typu

Každá operace může mít více implementací

**Použití**:

* **Přetěžování funkcí/metod**
  + Stejný název, liší se počtem nebo typem parametrů
  + V důsledku, funkce může dělat něco jinýho
* **Dědičnost** 
  + Rodičovská třída k definici (společné vlastnosti)
    - Odvozené třídy implementují odlišnost
  + 

**Důsledek**:

* Instance různých tříd v jedné hierarchii reagují stejně
* Potomek může zastoupit předka (může se chovat ale jinak)

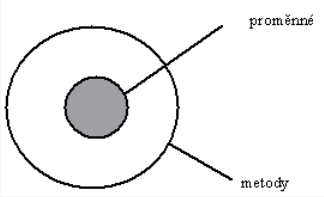
Text, letter

Description automatically generated

**Zapouzdření**

Umožňuje **před ostatními objekty zatajit vnitřní stav objektu**.

* Zamezuje jiným objektům změnu stavu objektu napřímo



* **Jádro = data | Slupka = metody**
  + Metody = komunikace s okolím, tímto se přistupuje k jádru/proměnám

Jak udělat zapouzdření?

* Skrytí detailů
* Nastavení přístupu k atributům
* Použití rozhraní pro komunikaci objektů
* Opatrné využívání dědičnosti