Třídy a objekty

alias objektové programování je zde

- Motivace: Chceme související údaje udržovat pohromadě,
- tak si na každou jednotku založíme objekt.
- Historické souvislosti: Potřeba vyvíjet delší programy. Assembler přestal být udržitelný,
- proto vzniklo strukturované programování,
- ovšem i to začalo být neudržitelné.
- Myšlenka objektového programování je dát objektům autonomii (a ony nám dají pokoj).

Třídy a objekty

podruhé a odjinud

- Svět kolem nás sestává z objektů,
- objekty jsou různých typů (přednášející, studenti, projektor,...)
- do značné míry se starají samy o sebe.
- Přesně takto to uděláme.

Třídy a objekty

a rozdíly mezi nimi

- Třída sdružuje (popisuje) objekty daného typu.
- Definujeme tedy třídy. Jako objekty vzniknou proměnné dotyčného typu.
- Třídu definujeme pomocí klíčového slova class
- a můžeme do ní ukládat proměnné (atributy) a funkce s atributy pracující (metody).
- Další prostředek dekompozice programu.

Třídy

příklad

```
class student:
    jmeno="Jan Novak"
    hodnoceni=4
    zapocet='N'
    def mam_zkousku(self):
        if self.hodnoceni==4:
            print("ne")
        else: print("ano")
    def udelej_zapocet(jasam,umim):
        if(umim>70):
            jasam.zapocet='Y'
```

Metody

jsou funkce uvězněné ve třídě

- Definují se podobně jako funkce, ale ve třídě.
- Mají první parametr, který se při volání sám doplní.
- Ten odkazuje na samotný objekt a jeho pomocí přistupujeme k atributům.
- Jak jej pojmenujeme, tak se bude jmenovat.
- V Pascalu `self`, v ostatních jazycích `this`, v Pythonu obvykle `self`, my se ale v létě nechceme zase vše přeučovat.

Objekty

konečně už budou

```
jan_novak=student()
if jan_novak.zapocet=='N':
    jan_novak.udelej_zapocet(int(input('Jak umim? ')))
print(jan_novak.mam_zkousku())
...
```

Objektové programování

a ideologie za ním

- Základní paradigmata (dědičnost, zapouzdřenost, polymorfismus) lépe uvidíme v létě.
- Jde o další prostředek dělení kódu. Kód tedy umíme dělit do funkcí a tříd.
- Další možností je dělení do souborů:
- `import module_name`

funkce.py:

```
def secti(a,b):
    return a+b
def odecti(a,b):
    return a-b
```

program.py:

```
import funkce
a=int(input('prvni cislo'))
b=int(input('druhe cislo'))
print(funkce.secti(a,b))
print('Jejich rozdil je ',funkce.odecti(a,b))
```

Příklad

který dává lepší smysl než ten na minulém slidu

```
import sys;
while True:
    print(ord(sys.stdin.read(1)))
```

- Vypisujeme ASCII-hodnoty napsaných znaků,
- funkce `read` přečte zadaný počet znaků ze vstupu (v našem případě jeden),
- funkce `ord` vrátí ordinální hodnotu znaku,
- funkce `read` je v modulu `sys`.
- Čtení vstupu po znacích použijeme, není-li vstup ve formátu vhodném pro načítání `input`.

Další možnosti importu

- `from funkce import secti`
- - importuje jen funkci `secti`.
- `import funkce as f`
- Teď k obsahu přistoupíme `f.secti...`
- `from funkce import secti as s`
- a funkci `secti` teď zavoláme `s(co,s_cim)`.

Veřejné a privátní atributy a metody

jsou v Pythonu dělány jmennou konvencí

- V Pythonu je vše veřejné.
- Chceme-li udělat položku privátní, zahájíme její název dvěma podtržítky.
- V ostatních jazycích je systematická opora pro ochranu objektu,
- uvidíme v létě v C#.