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`.

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())
...
```

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#.

Konstruktor

je funkce jmenující se `__init__`

- Objekt obvykle chceme netriviálně zinicializovat (při vzniku).
- K tomu slouží konstruktor, tedy funkce jménem `__init__`.
- Tato funkce je zavolána implicitně při tvorbě objektu:
- `jan_novak=student()`
- Platí stejná pravidla jako pro metody.

Příklad

konstruktoru

```
def __init__(self,jmeno,hodnoceni=4,zapocet='N'):
    self.jmeno=jmeno
    self.hodnoceni=hodnoceni
    self.zapocet=zapocet
    print('Zde konstruktor!')
```

Dědičnost

aneb třídy jako šablony

- Za jméno třídy do závorek napíšeme jméno rodiče.
- Pak zdědíme jeho atributy a metody:
- `class matfyzak(student):....`
- Další významný prostředek dekompozice programu (a eliminace zbytečného opakování kódu).
- Chceme-li přistoupit k rodiči: `super`,
- `super(matfyzak,self).__init__(...)`
- Při inicializaci potomka je třeba inicializovat i rodiče!

Příklad

dědičnosti

```
class matfyzak(student):
    __init__(self,jmeno='AB'
,hodnoceni=1,zapocet='Y',iq=300):
        print('Konstruktor matfyzacky')
        super(matfyzak,self).__init__(jmeno,hodnoceni,zapoce
        self.iq=iq
...
ab=matfyzak()
```

Statické a třídní metody

- Některé metody má smysl definovat pro třídu (ne každý objekt zvlášť),
- těm říkáme *statické*.
- V Pythonu se jim říká `classmethod`:
- `@classmethod def f(...):`
- volání: `jmenotridy.f(...);`
- `@staticmethod def f(...):`
- Statická metoda neví o atributech třídy (a nebere self).

Statické atributy

```
class a:
    x=1 #statický atribut!
    def f(self):
        self.x=10 #zalozi nestaticky atribut x
    @classmethod
    def sf(self):
        self.x=100 #modifikuje staticky atribut
```

Neexistuje-li nestatický atribut, objekty přistupují k atributu statickému!

Násobná dědičnost

je k vidění poměrně zřídka

- Můžeme chtít dědit od více předků:
- `class a:..., class b:...,`
- rodiče oddělíme čárkou:
- `class c(a,b):...`
- Tím zdědíme od všech jmenovaných předků.

Diamantový problém

já bych ho česky označil jinak... a byl by malér

- `class a:...,`
- `class b(a):..., class c(a):...,`
- `class d(b,c):...` – a třídu a zdědíme dvakrát.
- Python má důmyslný algoritmus co s tím (v každé verzi jiný).

Aplikace objektového programování

v podobě datové struktury

- Dynamické datové struktury jsou takové, do kterých lze prvky bez omezení přidávat a odebírat.
- Příklad: Spojové seznamy. Každý prvek nese hodnotu (hodnoty) a ukazatel na následníka.
- Ačkoliv existují předpřipravená řešení, my si ukážeme, jak spojové seznamy vytvořit sami.
- Využijeme objektové programování.

Příklad

spojového seznamu

```
class vagon:
    def __init__(self, hod, next=0):
        self.hod=hod
        self.next=next
class spojak:
    def __init__(self):
        self.zacatek=None
        self.konec=None
    def pridejZ(self,co):
        self.zacatek=vagon(co,self.zacatek)
        if self.konec==None: self.konec=self.zacatek
```