Přednáška 1

Python: Úvod do objektově orientovaného programovaní

Programovací paradigmata, základní koncepty OOP

155YOBP Objektové programování, zimní semestr 2022

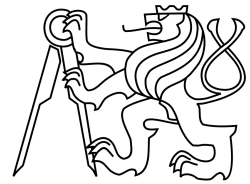
Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Martin Landa

martin.landa@fsv.cvut.cz

Fakulta stavební ČVUT v Praze
Katedra geomatiky

http://geo.fsv.cvut.cz/gwiki/155YOBP



Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Obsah přednášky

1 **Paradigmata**
- Imperativní
- Objektově orientované

2 **Koncepty OOP**
- Objekty
- Zapouzdření
- Kompozice
- Dědičnost

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Strojový kód | Paradigma nejnižší úrovně

- Posloupnost jednoduchých instrukcí prováděných procesorem počítače
- Zapsáno pomocí číselných kódů (sekvencí bitů ve dvojkové anebo šestnáctkové soustavě)
- Dvě části: kód instrukce a operandy (adresy definující data se kterými se má operace provést)
- Převod ze symbolického zápisu do strojového kódu zajišťuje assembler (překlad modulů) a linker (spojování modulů)
- Závislý na typu procesorů
- Pro člověka méně srozumitelný, postupně vznikla řada programovacích jazyků a paradigmat (stylů)

```
int 21h
11001101 00100001
```

https://en.wikipedia.org/wiki/Machine_code

Programovací paradigma | Nestrukturované paradigma

Imperativní (procedurální) programování

Výpočet definován posloupností příkazů, určuje přesný postup (algoritmus). Program je dán řadou proměnných, které v závislosti na vyhodnocení podmínek mění svůj stav (hodnotu).

Imperativní paradigma | Nestrukturované

- Lineární sekvence příkazů
- Skoky realizovány příkazem `GO TO` (čísla řádků, textová návěstí)
- Rané verze programovacího jazyka FORTRAN

```
IF (A) 20, 30, 40
20    Q = -(PI/2.0)
GO TO 50
30    Q = 0.0
GO TO 50
40    Q = PI/2.0
50
```


Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Programovací paradigma | Strukturované paradigma

Imperativní paradigma | Strukturované

- Nahrazuje příkazy skoku `GO TO` pomocí podmíněných cyklů (opakovat, dokud platí podmínka)
- Další strukturované instrukce, které se do sebe vnořují
- Typickými zástupci jazyk C anebo PASCAL

```
if (a < 0) {
    q = -(pi/2.0);
} else if (a > 0)
    q = pi/2.0;
} else {
    q = 0.0;
}
```

Programovací paradigmata

Komplexní problematika. Celá řada paradigmat – aspektově orientované, deklarativní, generické, metaprogramovaní, paralelní, atd.

https://en.wikipedia.org/wiki/Programming_paradigm

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Objektově orientované programování (OOP)

- Odlišuje se od imperativního programování
- Zdrojový kód přidružen k datům (tj. popisu jevu nebo vlastnosti pozorovaného objektu)
- Snadnější přenos kódu mezi projekty (abstrakce a zapouzdření)
- Snaha modelovat principy reálného světa v počítači pokud možno 1:1
- Snaha použít již vytvořené komponenty, podle potřeby je upravit nebo na základě nich sestavovat složitější objekty

https://en.wikipedia.org/wiki/Object-oriented_programming

Obsah přednášky

1 Paradigmata
Imperativní
Objektově orientované

2 Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Koncepty OOP

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Objekty

- Prvky modelované reality (data, funkčnost)
- Prvky seskupeny do entit (objektů)
- Objekty mají atributy (členské proměnné) — pamatují si svůj stav
- Objekty poskytují operace navenek — metody
- Metody jsou funkce, které mají přístup k atributům objektu
- Rozdělení objektů do tříd
- Objekt je instancí třídy

Abstrakce

- Objekty fungují jako „černé skříňky“
- Provádí činnost a komunikují bez toho, aby vyžadovali po okolí znalost, jak fungují uvnitř

Koncepty OOP

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Objekty

- Prvky modelované reality (data, funkčnost)
- Prvky seskupeny do entit (objektů)
- Objekty mají atributy (členské proměnné) — pamatují si svůj stav
- Objekty poskytují operace navenek — metody
- Metody jsou funkce, které mají přístup k atributům objektu
- Rozdělení objektů do tříd
- Objekt je instancí třídy

Abstrakce

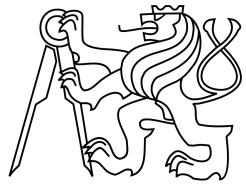
- Objekty fungují jako „černé skříňky“
- Provádí činnost a komunikují bez toho, aby vyžadovali po okolí znalost, jak fungují uvnitř

Koncepty OOP

Zapouzdření

- Nelze přistupovat k „vnitřnostem“ objektu přímo
- Objekt definuje rozhraní přes které s ním okolí (ostatní objekty) komunikuje
- Rozhraní je dáno modifikátory přístupu
  - *public, protected, private*

```
class Pes {
  private:
    bool bezim_za_mickem = false;
  public:
    void aport(bool stav) {
      bezim_za_mickem = stav;
    }
};

Pes p = Pes(); // p je instance objektu Pes
p.bezim_za_mickem = true; // NELZE
p.aport(true);            // LZE
```

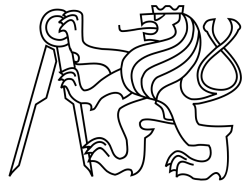
Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Zapouzdření a Python

- Jazyk Python nepoužívá modifikátory přístupu
- Koncept zapouzdření je „potlačen“

```
class Pes:
    bezim_za_mickem = False

    def aport(self, stav):
        self.bezim_za_mickem = stav

p = Pes() # p je instance objektu Pes
p.bezim_za_mickem = True # LZE
p.aport(True)            # LZE
```

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost



Možnosti zapouzdření a Python

- _ Protected
- __ Private

```
class Pes:
    __bezim_za_mickem = False

    def aport(self, ano):
        self.__bezim_za_mickem = ano
    def bezim(self):
        return self.__bezim_za_mickem

p = Pes()
p.bezim_za_mickem          # NELZE
p.__bezim_za_mickem        # NELZE
p.bezim_za_mickem = True   # LZE (!!)
p.bezim()                  # ?
p.aport(True)              # LZE
```

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Kompozice

- Objekt může obsahovat jiné objekty

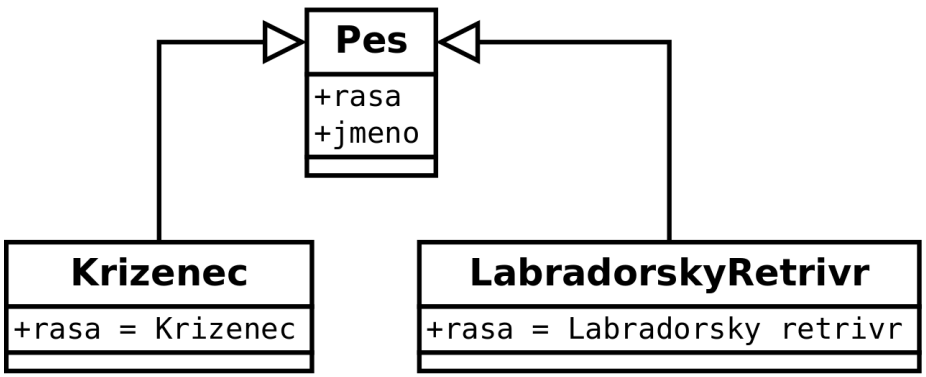
```
class Zaludek:
    plny = False

class Pes:
    zaludek = Zaludek()

    def hlad(self):
        return not self.zaludek.plny

p = Pes()
p.hlad()
```

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Dědičnost

- Objekty organizovány ve stromové struktuře
- Objekty mohou dědit vlastnosti objektů jiného druhu



Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Konstruktor

- Metoda s názvem `__init__()`
- Volá se ve chvíli vytvoření instance objektu
- Obdobně je definován destruktor `__del__()`, který se volá před odstraněním instance třídy z paměti počítače

```
class Pes:
    rasa = None
    def __init__(self, jmeno):
        self.jmeno = jmeno
        self.hmotnost_kg = -1

    def nastav_hmotnost(self, hmotnost):
        self.hmotnost_kg = hmotnost

class Krizenec(Pes):
    rasa = 'Kříženec'
class LabradorskyRetrivr(Pes):
    rasa = 'Labradorský retrívr'
```

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

```
g = Krizenec('Gulliver')
```

```
[In ] print(g.rasa, g.jmeno)
[Out] Kříženec Gulliver
```

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

```
b = LabradorskyRetrivr('Bára')
```

```
[In ] print(b.rasa, b.jmeno)
[Out] Labradorský Retrívr Bára
```

```
g.nastav_hmotnost(6)
b.nastav_hmotnost(22)

b.hmotnost_kg > g.hmotnost_kg # True
```

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Paradigmata
Imperativní
Objektově orientované
Koncepty OOP
Objekty
Zapouzdření
Kompozice
Dědičnost

Metoda super()

- Umožňuje přistupovat k atributům a metodám rodičovské třídy
- Typické využití volání konstruktoru rodičovské třídy

```
class Pes:
    def __init__(self, jmeno):
        self.jmeno = jmeno

class LabradorskyRetrivr(Pes):
    def __init__(self, jmeno, pp):
        super().__init__(jmeno)
        self.pp = pp

b = LabradorskyRetrivr('Bara', pp=False)
l = LabradorskyRetrivr('Lord', pp=True)
```