

IB111

# Programování a algoritmizace

Objektově orientované  
programování (OOP)

# OP a OOP

- Objekt – Kombinuje data a funkce a poskytuje určité rozhraní.
- OP = objektové programování
  - Vše musí být objekty
  - Např. Smalltalk, Ruby
  - Relativně málo rozšířené v dnešní praxi
- OOP = objektově orientované programování
  - Objekty jsou podporovány, ovšem ne všechno musí být nutně jen objekty.
  - Např. C++, Java, ...
  - Dnes velice rozšířený přístup

# Terminologie

- Třída
  - Typ objektu
- Objekt
  - Instance třídy
  - Struktura – atributy (data) a metody (funkce)
- Základní vlastnosti objektů v OOP
  - Abstrakce
  - Zapouzdření
  - Polymorfismus

# Základní paradigmatata OOP

- Datová abstrakce
  - Zavedeme vlastní datový typ
    - Struktura dat
    - Operace nad těmito daty
  - Oddělujeme rozhraní od implementace
  - Zapouzdření prvků třídy
- Dědičnost
  - Možnost rozšířit existující objekty
- Polymorfismus
  - Stejné rozhraní k různým objektům (díky zděděným společným vlastnostem).

# Zapouzdření

- Strukturální programování od sebe odděluje data a algoritmy (programový kód).
- OOP data zapouzdřuje v objektech a práce s nimi je umožněna přes definované rozhraní (pomocí metod objektu).
- Díky zapouzdření není třeba znát detaily implementace, ale stačí znát a používat rozhraní objektu.
- Díky zapouzdření je také možné změnit implementaci pokud použijeme stejné rozhraní.
- Díky „utajení“ je také možné vyhnout se chybám při náhodné nebo nesprávné přímé modifikaci dat.
- Umožňuje rozdělit úkoly v týmu programátorů.

# Zapouzdření – příklad v C

- ```
struct student {  
    int narozeni_rok, narozeni_mesic,  
    narozeni_den;  
    float studijni_prumer;  
    char jmeno[30];  
  
    int vypocitej_vek();  
};
```

# Zapouzdření – příklad v Pythonu

- `class student:`  
    `narozeni_rok, narozeni_mesic,`  
    `narozeni_den = 0,0,0`  
    `studijni_prumer, jmeno = 0.0, ""`  
  
    `def vypocitej_vek(self):`  
        `...`

# Zapouzdření

- Data a metody ve třídě mohou být několika typů podle jejich „viditelnosti“
  - **public** – k datům a proměnným je volný přístup
    - Správný OOP program by měl ponechat public pouze metody, ne data.
  - **private** – metody a data nejsou přístupná z jiných částí kódu než z metod této třídy
  - **protected** – metody a data nejsou volně přístupné, ale jsou přístupné potomkům této třídy.



# Zapouzdření - příklad

- ePeněženka – rozhraní třídy
- ```
class penezenka {  
private:  
    int koruny, halire;  
public:  
    penezenka () { koruny = halire = 0; };  
  
    void nabij (int k);  
    BOOL zaplat (int k, int h);  
    void vypis_aktualni_stav();  
};
```

# Zapouzdření - příklad

- Implementace
- `void penezenka::nabij (int k)`

```
{  
    koruny+=k;  
}
```
- `BOOL penezenka::zaplat (int k, int h)`

```
{  
    long zbytek = koruny*100+halire - (k*100+h);  
    if(zbytek<0) return FALSE;  
    koruny = zbytek /100; halire = zbytek%100;  
    return TRUE;  
}
```

# Zapouzdření - příklad

- Použití:
- `penezenka moje, cizi;`

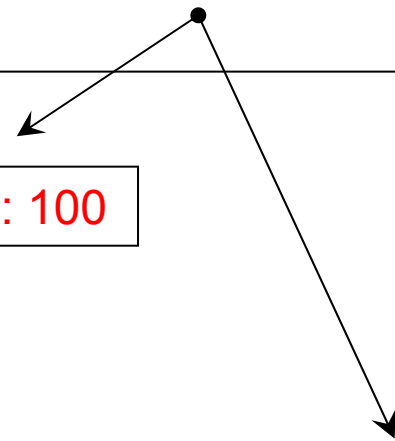
```
moje.nabij(100);  
moje.zaplat(2,50);  
moje.vypis_aktualni_stav();
```

```
cizi.nabij(5);  
cizi.vypis_aktualni_stav();
```

```
void penezenka::nabij (int k)  
{  
    koruny+=k;  
}
```

moje: 100

cizi: 5



# Zapouzdření - Python

```
class MyClass:  
    i = 12345  
    def f(self):  
        return 'hello world'
```

- Python nezná „private“ členy třídy
- Domluva: co začíná znakem „\_“ je neveřejné
- Specialitka: co začíná „\_\_“ tak se upraví

# Python - příklad

7c.py - C:/Documents and Sett

File Edit Format Run Options

```
class nic:  
    def __a(self):  
        return 2
```

```
>>> n=nic()
```

```
>>> n.__a()
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#16>", line 1, in <module>
```

```
    n.__a()
```

```
AttributeError: nic instance has no attribute '__a'
```

```
>>> n._nic__a()
```

```
2
```

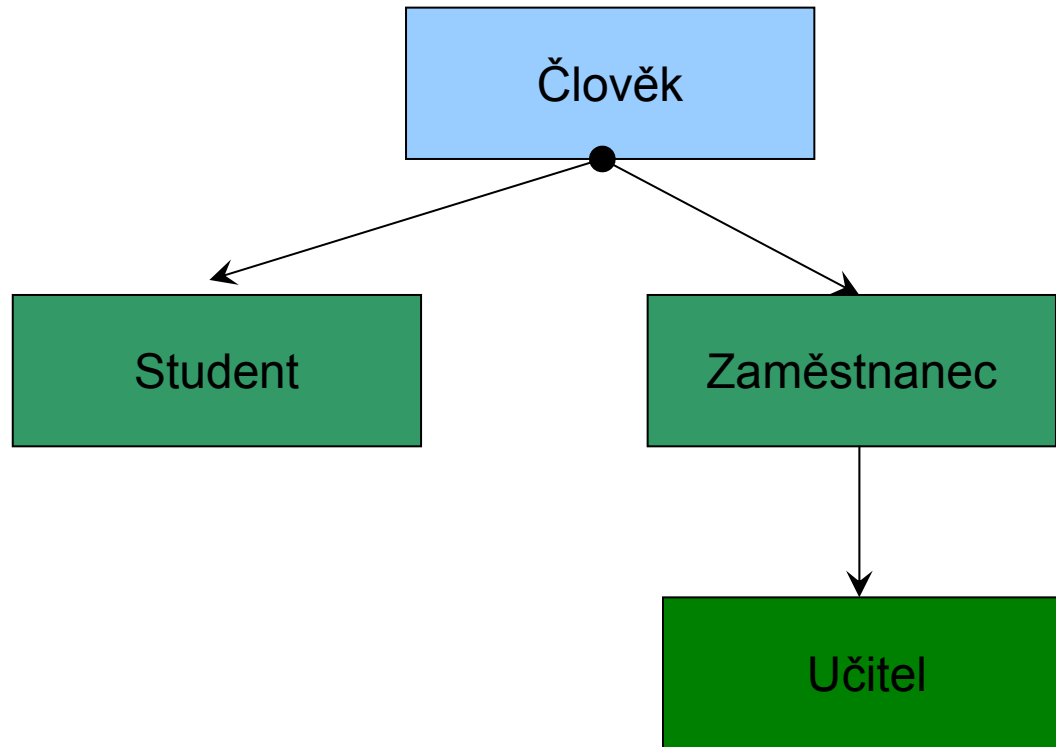
# Dědičnost

- Dědičnost umožňuje rozšiřování funkčnosti třídy.
- Potomek je nová třída založená na třídě původní (rodičovské). Potomek přebírá vlastnosti rodiče a může být dál rozšiřován (nová data a metody).

# Dědičnost - příklad

- Základní objekt (rodič) bude *člověk*.
- Potomky budou *student*, *zaměstnanec* a *učitel*.
- Každý *člověk* má základní atributy jako jméno, příjmení, datum narození a adresu.
- *Studenti* mají navíc studijní průměr, a jsou zapsáni v určitém ročníku.
- *Zaměstnanci* mají plat a ten je jim posílán na nějaký bankovní účet.
- *Učitelé* jsou zaměstnanci, ale mají konzultační hodiny a učí nějaké předměty

# Dědičnost - příklad





# Příklad v Pythonu

74 st.py - C:\Documents and Settings\Administrator\Plocha\st.py

File Edit Format Run Options Windows Help

```
class clovek:
    jmeno, prijmeni = "", ""
    narozeni_den, narozeni_mesic, narozeni_rok = 0, 0, 0
    adresa = ""

    def vypocitej_vek(self):
        return 20

class student (clovek):
    rocnik=0
    prumer=0.0

class zamestnanec (clovek):
    plat=0
    iban=""

class ucitel (zamestnanec):
    konzultacky = ""
    seznam_predmetu = ""
```

```
>>>
>>> u=ucitel()
>>> u.vypocitej_vek()
20
>>>
```

# Dědičnost – příklad v C++

```
class clovek {
public:
    char jmeno[20],prijmeni[30];
    int narozeni_rok, narozeni_masic, narozeni_den;
    char adresa [100];
    int vypocitej_vek(); };
class student : public clovek {
public:
    int rocnik;
    float prumer; };
class zamestnanec : public clovek {
public:
    int plat;
    char IBAN [20]; };
class ucitel : public zamestnanec {
public:
    char konzultacky[50];
    list_of_subjects *uci; };
```

# Dědičnost – příklad v C++

```
clovek c;
```

```
zamestnanec z ;
```

```
ucitel u;
```

```
int v1= c.vypocitej_vek();
```

```
int v2= z.vypocitej_vek();
```

```
c = u;           // i učitel je člověk
```

```
z = c;           // NELZE – ne každý člověk  
                  je zaměstnanec
```

# Polymorfismus

- Polymorfismus = jednotné zacházení s různými (polymorfními) objekty mající některé společné zděděné vlastnosti.
- Polymorfismus v praxi znamená, že je možné mít různé třídy s metodami se stejnými parametry. Mohu tak odlišit chování potomků.
  - Pozdní vazba – late binding
- Polymorfismus umožňuje mít stejně pojmenované metody s různými parametry.
  - Parametry se mohou lišit v počtu a/nebo typu.
  - Mluvíme o tzv. přetížení metod (funkcí).

# Příklad v Pythonu

```
74 st.py - C:\Documents and Settings\Administrator\Plocha\st.py
File Edit Format Run Options Windows Help

class clovek:
    jmeno, prijmeni = "", ""
    narozeni_den, narozeni_mesic, narozeni_rok = 0, 0, 0
    adresa = ""

    def sazba(self):
        return 5

class student (clovek):
    rocnik=0
    prumer=0.0
    def sazba(self):
        return 6

class zamestnanec (clovek):
    plat=0
    iban=""
    def sazba(self):
        return 8

class ucitel (zamestnanec):
    konzultacky = ""
    seznam_predmetu = ""
    def sazba(self):
        return 9
```

```
>>> c=clovek()
>>> c.sazba()
5
>>>
>>> u=ucitel()
>>> u.sazba()
9
>>> |
```

# Polymorfismus - příklad

- `void penezenka::zaplat (int k, int h)`

```
{  
    long zbytek = koruny*100+halire - (k*100+h);  
    if(zbytek<0) return FALSE;  
    koruny = zbytek /100; halire = zbytek%100;  
    return TRUE;  
}
```
- `void penezenka::zaplat (int k)`

```
{  
    long zbytek = koruny - k;  
    if(zbytek<0) return FALSE;  
    koruny = zbytek;  
    return TRUE;  
}
```
- `void penezenka::zaplat (void)`

```
{  
    long zbytek = koruny - 10;  
    if(zbytek<0) return FALSE;  
    koruny = zbytek;  
    return TRUE;  
}
```

# Polymorfismus - příklad

- penezenka moje;

```
moje.nabij(100);
```

```
moje.zaplat(10,0);
```

```
moje.zaplat(10);
```

```
moje.zaplat();
```

```
moje.vypis_aktualni_stav();
```

# Oprátory new, delete v C++

- Vytvoření objektu a jeho rušení pomocí dynamické alokace paměti.
- `penezenka *moje = new penezenka;`

`moje -> nabij(100);`

`moje -> zaplat(5,50);`

`delete moje;`



# Konstruktory, destruktory

- Metody, které jsou (automaticky) volány při vytváření/rušení instance objektu.
- Vhodné pro
  - Inicializaci proměnných
  - Alokaci/dealokaci paměti

# Konstruktory - příklad

- ```
class student {  
private:  
    float prumer;  
    char *jmeno;  
public:  
    student () {prumer=0.0; jmeno=NULL; }  
    student(float p,char *j) { prumer=p;  
        jmeno = (char*)malloc(strlen(j)+1);  
        if(jmeno)strcpy(jmeno,j); }  
    ~student() { if(jmeno) free(jmeno); }  
    ...  
};
```

# Příklad v Pythonu

7% complex.py - C:/Documents and Settings/Administrator/Plod

File Edit Format Run Options Windows Help

```
class Complex:
    def __init__(self, real, imag):
        self.r = real
        self.i = imag
    def __del__(self):
        print "Mazu objekt ..."
```

```
c = Complex(1,-1)
```

```
print c.r
```

```
print c.i
```

```
del c
```

```
|
```

```
>>>
```

```
1
```

```
-1
```

```
Mazu objekt ...
```

```
>>>
```

# Přetěžování operátorů v C++

- Mohu přetížit (změnit) běžné operátory
  - Např. +, -, +=
  - Přetížit lze téměř všechny operátory
  - Nelze přetížit ., ?::, sizeof, ::, .\*
- Přetížení operátorů velice usnadňuje používání tříd/objektů programátory

# Přetěžování operátorů - příklad

```
class komplex {  
public:  
    float Re, Im;  
  
    komplex (float r, float i) { Re=r; Im=i; };  
    komplex komplex::operator+=(komplex &b)  
    {  
        Re+=b.Re; Im+=b.Im;  
        return *this;  
    };  
    komplex komplex::operator+(komplex &b)  
    {  
        return komplex (Re + b.Re, Im + b.Im);  
    };  
};
```

```
komplex a(1,0), b(2,0);  
komplex c = a + b;  
c+= komplex(1,5);
```

# Přetěžování operátorů v Pythonu

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)
object.__xor__(self, other)
object.__or__(self, other)
```

# Přetěžování operátorů v Pythonu

```
74 complex.py - C:\Documents and Settings\Administrator\Plocha\complex.py
File Edit Format Run Options Windows Help

class Complex:
    def __init__(self, real, imag):
        self.r = real
        self.i = imag

    def __del__(self):
        print "Mazu objekt ..."

    def __add__(self,b):
        return Complex(self.r + b.r, self.i + b.i)

c = Complex(1,-1)
print c.r
print c.i

a=Complex(1,1)
b=Complex(2,3)

d=a+b

print d.r
print d.i

del c
```

```
>>>
1
-1
3
4
Mazu objekt ...
>>> |
```