IB111 Programování a algoritmizace

Objektově orientované programování (OOP)

Struktury v C - opakování

- struct student {
  int vek;
  float studijni_prumer;
  char jmeno[30];
  } st, *uk_st;
- st.vek=22; st.studijni_prumer=1.12;
  strcpy (st.jmeno, “Jan Kos”);
- uk_st = &st; uk_st ->vek=23;
- struct student prvaci[100]; prvaci[0].vek=25;

OP a OOP

- Objekt – podobný na strukturu v C, kombinuje však data a algoritmy 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 paradigmata 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

- struct student {
  int narozeni_rok, narozeni_mesic,
  narozeni_den;
  float studijni_prumer;
  char jmeno[30];

  int vypocitej_vek();
  };

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 něž 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

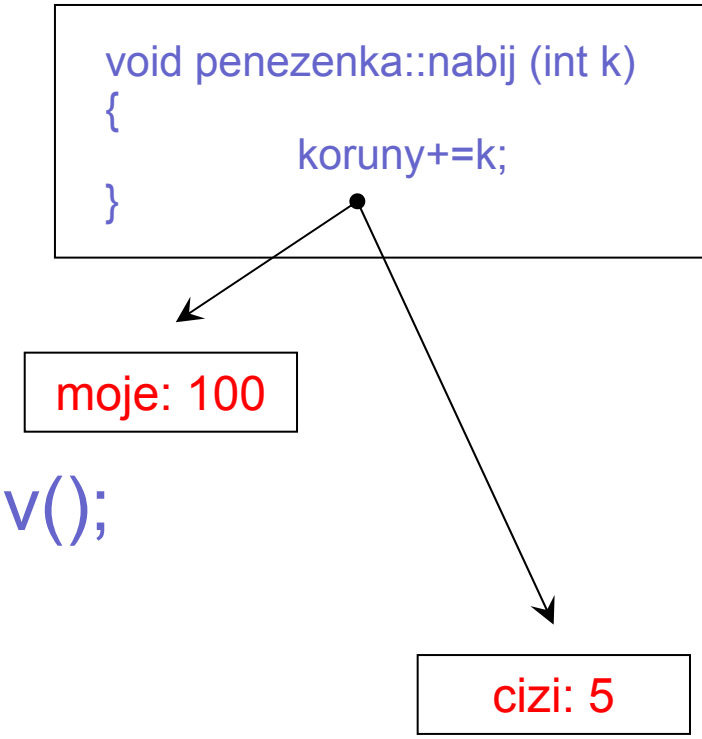
- Implementace
- void penezenka::nabij (int k)
  {
  koruny+=k;
  }
- BOOL penezenka::zaplat (int k, int h)
  {
  long zbytek = koruny*100+halire – (k*100+h);
  if(zbytek<0) return FALSE;
  koruny = zbytek /100; halire = zbytek%100;
  return TRUE;
  }

Zapouzdření - příklad

- Použití
- penezenka moje, cizi;

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

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



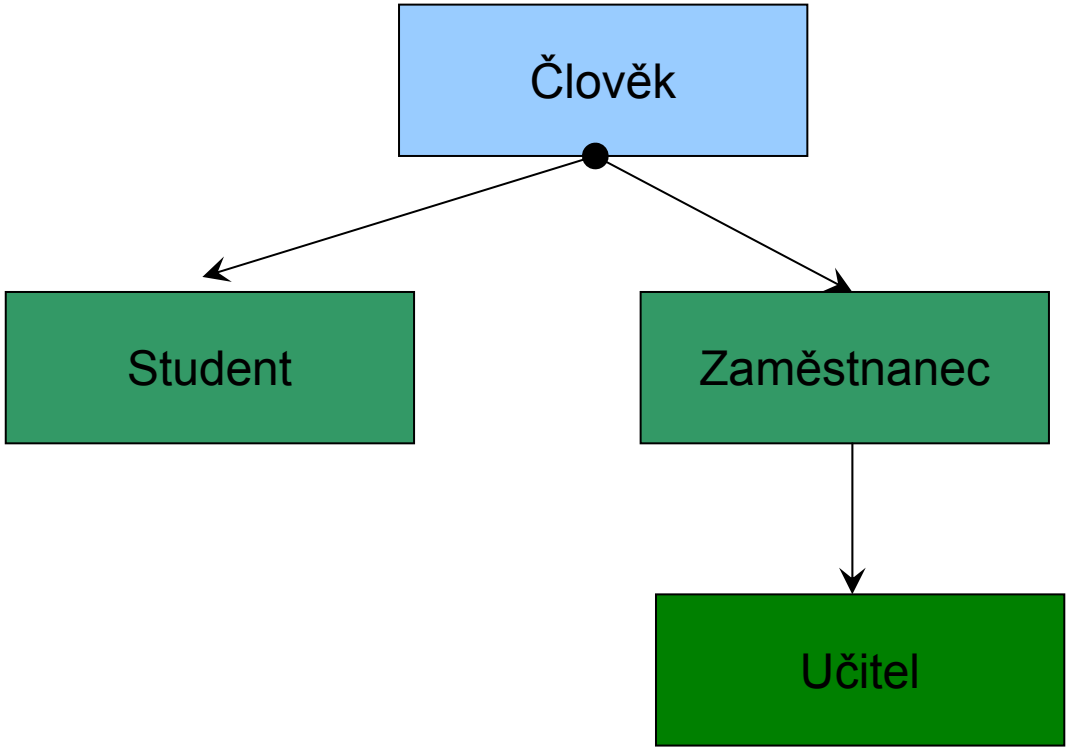
Dědičnost

- 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



Dědičnost - příklad

```
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 - příklady

```
clovek jan;
zamestnanec jirka;
ucitel petr;

int v1= jan.vypocitej_vek();
int v2= jirka.vypocitej_vek();

jan = petr;      // i učitel je člověk
jirka = jan;     // 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 třídy s různými parametry.
  - Parametry se mohou lišit v počtu a/nebo typu.
  - Mluvíme o tzv. přetížení metod (funkcí).

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

- 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

Kontruktory - 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); }
        …
};
```

Přetěžování operátorů

- 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);
```

Příští přednáška

- Následuje cvičení v B311 v 14:00
- Poslední přednáška 15.12.2009 v B410 od 12:00