Základy programování v C++ 20. cvičení Letmý úvod do objektově orientovaného programování (v C++) I.

Zuzana Petříčková

4. prosince 2019

Přehled

Letmý úvod do objektového programování (v C++)

Softwarový objekt

- model nějaké části reálného světa (např. **Pepa**)
- objekty řadíme do **tříd** (např. **člověk, zaměstnanec, zlomek,...**)

Základní myšlenka

- **Datový typ** (jak ho známe, např. **int**)
  - je určen množinou přípustných hodnot a množinou operací, které nad touto množinou lze provádět
- **Objektový datový typ** (**třída**)
  - definuje množinu hodnot
  - navíc definuje i operace nad touto množinou

Letmý úvod do objektového programování (v C++)

Datový typ struktura

- přímo obsahuje (definuje) datové složky
- (bokem) definujeme podprogramy (funkce), které s proměnnými daného typu pracují

Objektový datový typ (třída)

- přímo obsahuje (definuje) datové složky
- přímo obsahuje (definuje) i operace nad objekty dané třídy

Terminologie

- **třída (class)** ... objektový datový typ
- **instance (objekt, object)** ... proměnná objektového typu

Složky třídy

- **atribut** ... datová složka
- **metoda** ... funkce (funkční složka)
  - **konstruktor** ... postará se o vytvoření a inicializaci nové instance
  - **destruktor** ... postará se o zrušení instance

1. **instanční** ... metoda pro práci s jednotlivými instancemi
2. **třídní** ... metoda pro práci s třídou jako celek

stručně:

- na rozdíl od struktury definujeme u třídy zároveň **atributy** i **metody**
- **třída** ale přináší další nové možnosti

Příklad ... třída jako rozšíření struktury o metody

```
struct Zamestnanec
{
    string jmeno;
    string prijmeni;
    void vypis()
    {
        cout << jmeno << " " << prijmeni << endl;
    }
};
```

```
int main()
{
    Zamestnanec z;
    z.jmeno = "Josef";
    z.prijmeni="Novak";
    z.vypis();

    Zamestnanec *pz;
    pz = new Zamestnanec;
    pz->jmeno = z.jmeno;
    pz->prijmeni = "Svoboda";
    pz->vypis();
    delete pz;
}
```

Příklad ... třída jako rozšíření struktury o metody

```
struct Zamestnanec
{
    string jmeno; string prijmeni;
    string jmenoAPrijmeni()
    {
        return jmeno + " " + prijmeni;
    }
    void vypis()
    {
        //cout << jmeno << " " << prijmeni << endl;
        //cout << this->jmeno << " " << this->prijmeni << endl;
        //cout << jmenoAPrijmeni()<<endl;
        cout << this->jmenoAPrijmeni()<<endl;
    }
};
```

- metoda může pracovat s atributy objektu (měnit je) a volat jeho další metody
- přístup k atributům i jiným metodám instance přímo nebo pomocí ukazatele **this**

Nové možnosti, které umožňují objektové datové typy

Zapouzdření (ukrývání implementace, encapsulation)

- datové složky a metody definujeme ,,na jednom místě"
- můžeme omezit přístup k určitým datovým složkám
  - **pravidlo**: k datovým složkám přistupovat pouze prostřednictvím metod

Dědění (dědičnost, inheritance)

- možnost od jedné třídy (**předek**) odvodit jinou třídu (**potomek**)
- potomek může zastoupit předka
- potomek **zdědí** metody a atributy předka, některé metody může překrýt, další metody a atributy může doplnit

Polymorfismus (mnohotvárnost)

- jednotné rozhraní pro práci s různými typy objektů
- práce s instancemi neznámého typu (předek? / potomek?), určení typu instance za běhu programu

Deklarace objektového typu (prozatím bez dědění)

- klíčová slova **class, struct**

```
class identifikator // struct identifikator
{
    <seznam sekci>
}
```

- tělo třídy je rozděleno do sekcí s různou specifikací přístupu (**public, protected, private**)
- **sekce:**

```
<specifikace pristupu>
<seznam atributu>
<seznam metod>
```

Deklarace objektového typu (prozatím bez dědění)

Přístup ke složkám instancí

- **public** ... veřejné složky (mohou je používat všechny části programu)
- **private** ... soukromé složky (přístupné pouze metodám třídy a tzv. **přátelům**)
- **protected** ... chráněné složky (přístupné pouze metodám třídy, přátelům a potomkům)

Rozdíl mezi struct a class

- **struct** ... přístup je implicitně nastaven jako **public**
- **class** ... přístup je implicitně nastaven jako **private**

Příklad

```
class Zamestnanec
{
  public:
    void vypis()
    {
     // cout << jmeno << " " << prijmeni
     //      << " " << plat << endl;
         cout << this->jmeno << " " << this->prijmeni
              << " " << this->plat << endl;
    }
   private:
    string jmeno = "";
    string prijmeni = "";;
    unsigned int plat = 0;
};
```

Deklarace objektového typu

definiční deklarace metod mohou být

1. přímo v těle třídy
2. mimo tělo třídy
   - v těle třídy je jen informativní deklarace
   - definiční deklarace je v tomto případě kvantifikovaná názvem třídy (jinak by překladač nevědel, ke které třídě metoda patří)
   - pro třídy s mnoha složkami je to přehlednější varianta

```
void Zamestnanec::vypis()
{
  cout << jmeno << ' ' << prijmeni << ' ' << plat << endl;
}
```

Příklad ... pokračování

```
class Zamestnanec
{
  public:
    void vypis();
  private:
    string jmeno = "";
    string prijmeni = "";;
    unsigned int plat = 0;
};
void Zamestnanec::vypis()
{
  cout << jmeno << " " << prijmeni
       << " " << plat << endl;
}
```

Zapouzdření

Zapouzdření (encapsulation)

- datové složky a metody definujeme ,,na jednom místě"
- můžeme omezit přístup k určitým datovým složkám
- **v čistém OOP platí pravidlo**: k datovým složkám přistupovat důsledně prostřednictvím (přístupových) metod
- **proč?**:
  - ukrytí implementace
  - bezpečnost (třída má svoje data pod kontrolou)
  - úspora pozdější práce (snadno mohu datovou reprezentaci časem změnit)

Zapouzdření

Další pojmy

- **Rozhraní třídy** (interface)
  - seznam **veřejných** metod a **veřejných** datových složek spolu s jejich popisem
- **Přístupová metoda** (accessor)
  - metoda, která nastavuje nebo vrací hodnotu atributu (**setter, getter**)
- **Vlastnost** (feature)
  - atribut doplněný o tzv. přístupové metody

Správné zapouzdření

→ pokud změním implementaci třídy a zachovám přitom její rozhraní, nemusím měnit jiné části programu

Zapouzdření ... příklad

```
class Zamestnanec
{
    string jmeno;
    unsigned int plat;
public:
    void nastavJmeno(const string& jmeno) // setName
    {
        this->jmeno = jmeno;
    }
    float vratJmeno()  // getName
    {
        return this->Jmeno;
    }
    void nastavPlat(unsigned int plat)
    {
        if ((plat > 10000)&& (plat < 100000))
            this->plat = plat;
    }
    float vratPlat() // getSalary
    {
        return this->plat;
    }
```

Zapouzdření ... příklad

```
int main()
{
    Zamestnanec z;
    z.nastavJmeno("Jan Novak");
    z.nastavPlat(20000);
    cout << z.vratJmeno() << " "
         << z.vratPlat() << endl;
    return 0;
}
```

Konstantní atributy a metody

Konstantní metoda

- označení metody, která nemění datové složky instance (správně by tak měly být označeny všechny gettery):

```
void Zamestnanec::vratJmeno() const
{
    return jmeno;
}
```

Pravidla

- z konstantní metody lze volat jen konstantní metody
- pro konstantní instance lze volat pouze konstantní metody

Přístupová práva ... doplnění

Spřátelená funkce

- není to metoda třídy, ale má při přístupu ke složkám stejná práva jako metody:
  - může přistupovat k soukromým i chráněným složkám třídy
- deklarace **friend** kdekoliv v těle třídy (specifikace přístupu zde nehraje roli):

```
friend informativni_deklarace_funkce;
friend definicni_deklarace_funkce;
friend class jmeno_tridy; // vsechny metody tridy jsou spratelene
```

Spřátelená funkce ... příklad

```
class Zamestnanec
{
    string jmeno;
    unsigned int plat;
public:
    ...
    friend void vypis(Zamestnanec &z);
    friend unsigned int premie(Zamestnanec &z)
    {
       return z.plat *0.1;
    }
};

void vypis(Zamestnanec &z)
{
   cout << z.jmeno << " " << z.plat << endl;
}
```

Spřátelená funkce ... příklad

```
int main()
{
    Zamestnanec z;
    z.nastavJmeno("Jan Novak");
    z.nastavPlat(20000);
    cout << z.vratJmeno() << " "
         << z.vratPlat() << endl;
    vypis(z);
    cout << premie(z) << endl;
    return 0;
}
```

Příklad na procvičení I : Zlomek

- Přeměňte strukturu Zlomek z 11. cvičení na třídu se soukromými atributy **citatel** a **jmenovatel**.
  - Implementujte jednotlivé přístupové metody (gettery a settery) pro Zlomek, vytvorte i setter se dvěma parametry (citatel a jmenovatel)
    - Settery převedou zlomek do základního tvaru
    - Gettery budou konstantní metody.
  - Původní funkce nad jedním Zlomkem (**vypis** a **zakladniTvar**) změňte na metody (první z nich bude veřejná a konstantní, druhá bude soukromá)
  - Funkce nad dvěma Zlomky změňte tak, aby správně a efektivně využívaly přístupové metody.
  - Zkuste implementovat varianty výpočtů nad zlomky ve tvaru:

```
void Zlomek::pricti(const Zlomek &a);// pricte zlomek k akt. insta
void Zlomek::vynasob(const Zlomek &a);// vynasobi aktualni instanc
...                                   // zlomkem
```