ZPRO 11. cvičení

Základní myšlenka

- Datový typ (např. int)
  - Je určen množinou přípustných hodnot a množinou operací, které nad touto množinou lze provádět
- Objektový datový typ (třída)
  - Definuje množinu hodnot
  - Definuje operace nad touto množinou

Úvod do objektového programování – datové typy

- (neobjektový) Datový typ struktura
  - Přímo obsahuje (definuje) datové složky
  - Definujeme funkce (bokem), které s proměnnými (atributy) daného datového typu pracují
- Objektový datový typ
  - Přímo obsahuje (definuje) datové složky
  - Přímo obsahuje (definuje) i operace nad objekty dané třídy

Terminologie

- Třída (class) – objektový datový typ
- Instance == objekt – proměnná objektového typu

- Složky tříd:
  - Atributy – datové složky třídy
  - Metoda – funkce třídy
    - Speciální metody:
    - **Konstruktor** – metoda, která se postará o vytvoření instance
    - **Destruktor** – metoda, která se postará o zrušení instance

Možnosti, které umožňují objektové datové typy

- Zapouzdření (encapsulation): K datovým složkám a dalším implementačním detailům mají přístup pouze metody daného objektového typu.
- Dědění (specializace): Odvození nové třídy jako specializace, tedy upřesnění existující třídy. Mohou přibýt nové metody a datové složky, může se změnit význam (implementace) některých metod.

  – Předek, potomek; bázová (základní) třída, odvozená třída; rodičovská, dceřinná třída.
- Polymorfismus: Instance odvozených tříd lze použít tak, kde je očekávána instance předka.

Příklad: Třída jako rozšíření struktury o metody

Deklarace objektového datového typu

deklarace třídy:
klíč identifikátor specifikace_předků$_{\text{nep}}$ { *tělo* } ;

Zde klíč je **class** nebo **struct**.

specifikace_předků:
: *seznam_předků*

seznam předků:
přístup$_{\text{nep}}$ *předek*
přístup$_{\text{nep}}$ *předek*, *seznam_předků*

Deklarace objektového datového typu

```
class Třída : Předek1, Předek2
{
    // Tělo třídy
    // Atributy - datové složky třídy
    int atribut1;
    string atribut2;
    // Metody - funkce třídy
    void výpis();
};
```

Sekce – specifikace přístupu

- Přístup ke složkám instancí
  - **public** – veřejné složky -mohou je používat všechny části programu
  - **private** – soukromé složky – mohou je používat pouze metody třídy a přátelé
  - **protected** – chráněné složky – přístupné pouze metodám, přátelům a potomkům
- Rozdíl mezi struct a class
  - **struct** - přístup je implicitně nastaven jako **public**
  - **class** – přístup je implicitně nastaven jako **private**

```
class Třída : Předek1, Předek2
{
private:
    // Tělo třídy
    // Atributy - datové složky třídy
    int atribut1;
    string atribut2;
public:
    // Metody - funkce třídy
    void výpis();
};
```

Definiční deklarace

- Přímo v těle třídy
- Mimo tělo třídy
  - V těle třídy pouze informativní deklarace
  - Definiční deklarace je v tomto případě kvantifikovaná názvem třídy

Definiční deklarace

```
class Třída : Předek1, Předek2
{
private:
    int atribut1;
    string atribut2;
public:
    void výpis()
    {
        cout << atribut1 << endl;
        cout << atribut2 << endl;
    }
};
```

```
class Třída : Předek1, Předek2
{
private:
    int atribut1;
    string atribut2;
public:
    void výpis();
};


void Třída::výpis()
{
    cout << atribut1 << endl;
    cout << atribut2 << endl;
}
```

Zapouzdření - encapsulation

- Datové složky a metody definujeme na jednom místě
- Můžeme omezit přístup k některým datovým složkám a metodám
- V čistém OOP platí pravidlo: k datovým složkám přistupovat důsledně prostřednictvím přístupových metod
- Důvody:
  - Bezpečnost (třída má svoje data pod kontrolou)
  - Ukrytí implementace
  - Úspora pozdější práce

Další terminologie

- Rozhraní třídy - Interface
  - Seznam veřejných metod a veřejných datových složek
- Přístupové metody - accessor
  - Metoda, která nastavuje (**setter**) a vrací hodnotu (**getter**) atributu
- Vlastnost
  - Atribut doplněný o přístupové metody

Konstantní metoda

- Metoda, která nemění hodnoty atributů
- Pravidla:
  - Z konstantní metody lze volat jen konstantní metody
  - Pro konstantní instance lze volat jen konstantní metody

```
class Třída : Předek1, Předek2
{
private:
    int x;
public:
    int getX() const;
};

int Třída::getX() const
{
    return x;
}
```

Zpřátelené funkce a třídy

- Funkce (třída), která má při přístupu ke složkám třídy stejná práva jako metody
- Deklarace **friend** kdekoli v tělě třídy

```
class Třída
{
private:
    int x;
public:
    int getX() const;

    friend class Přítel;
    friend void ZpřátelenáFunkce(const Třída & t);
};

void ZpřátelenáFunkce(const Třída & t)
{
    cout << t.x << endl;
}
```

Speciální metody

- Konstruktor
  - Inicializace atributů
  - Vytvoření (alokace) dynamických datových složek
- Destruktor
  - Zrušení (dealokace) dynamických datových složek
  - Zavření proudů apod.

Konstruktor

- Metoda, která slouží k vytvoření a inicializaci instance
- Konstruktor nelze volat přímo, je volán automaticky při:
  - Vytvoření instance (staticky nebo dynamicky (new))
  - Při předávání parametrů objektového typu hodnotou
  - Při konverzích
- Jmenuje se stejně jako jméno třídy
- Třída může mít několik konstruktorů

Destruktor

- Metoda, která slouží k zrušení instance
- Destruktor je volán automaticky při zániku instance:
  - U lokální instance – je volán na konci bloku
  - U globální a statické instance – je volán po skončení funkce main()
  - U dynamické instance – operátor delete nejprve zavolá destruktor a pak uvolní paměť, kterou instance zabírala
- Jmenuje se stejně jako třída navíc před jménem má znak ~
- Každá třída může mít pouze jeden destruktor

Příklad Chytré Pole objektově

Kopírovací konstruktor – copy constructor

- Specifický konstruktor, slouží k vytvoření kopie instance
- Pokud nedeklarujeme kopírovací konstruktor, překladač vytvoří vlastní tzv. implicitní :
  - Neobjektové atributy přenese („překopíruje“)
  - Objektové složky překopíruje pomocí jejich kopírovacích konstruktorů
  - Vytváří tzv. **mělkou** kopii (v případě dynamických složek se překopíruje jen ukazetel!)

Kopírovací konstruktor – copy constructor

- Pokud instance obsahuje dynamicky alokovanou paměť, nebude implicitní konstruktor pracovat správně
- Potřebujeme vlastní **kopírovací konstruktor**
- Ten se postará přidělení odpovídající dynamické paměti a překopírování obsahu
- Kopírováním získáme tzv. **hlubokou kopii**
- Syntaxe:
  - Konstruktor, který má jako parametr referenci na instance stejné třídy