Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 04.07. 2002
Url: http://www.builder.cz/art/cpp/oopend.html

Dokončení seriálu objektově orientované programování v C++

Dnešním článkem ukončím seriál Objektově orientované programování v C++. Nejprve se ale musím přiznat, že jsem v úvodu seriálu nějak pozapomněl na dvě dost podstatné klíčové slova. Jedná se o `static` a `mutable`. Jedná se vlastně o základy jazyka C++. Správně jsem se o nich měl zmínit tak maximálně v pátém článku seriálu, nikoliv v posledním díle.

Klíčové slovo mutable

Jak jsem již dříve napsal, v C++ lze metodu označit klíčovým slovem `const`. Tím říkáme překladači, že metoda nijak nemění vnitřní stav objektu. Lze ji tedy zavolat na konstantní objekt. Znamená to, že v těle takové metody nesmí dojít ke změně atributů objektu. Změníme-li atributy objektu v těle "konstantní" metody, překladač ohlásí syntaktickou chybu. Přece ale existují atributy, které lze měnit konstantní metodou. Takový atribut musí být označen klíčovým slovem `mutable`. Příklad:

```
#include<iostream>
using namespace std;

class Trida
{
    private:
        mutable int MAtribut;
        int Atribut;
    public:
        Trida() : MAtribut(0), Atribut(0) {}
        void zvetsi() { MAtribut++; Atribut++; }
        int vrat() const;
        int vratMutableAtribut() const { return ++MAtribut; }
};

int Trida::vrat() const
{
    MAtribut++;
    // Odkomentování následujícího řádku je syntaktická chyba
```

```
    // Atribut++;
    return Atribut;
}

int main(int, char**)
{
    Trida objekt1;
    objekt1.zvetsi();
    objekt1.zvetsi();
    cout << objekt1.vrat() << endl;
    cout << objekt1.vratMutableAtribut() << endl;
    const Trida objekt2;
    cout << objekt2.vrat() << endl;
    cout << objekt2.vratMutableAtribut() << endl;
    return 0;
}
```

Není dobré klíčové slovo `mutable` psát neuváženě jen proto aby šlo obejít omezení konstantních objektů. Slovo `mutable` slouží hlavně k označení atributů, které přímo nesouvisejí s vnitřním stavem objektu. Lze je tedy měnit i v případě, že objekt je konstantní.

Klíčové slovo static

Z jazyka C "zdědil" jazyk C++ klíčové slovo `static`, které lze použít u globálních nebo lokálních proměnných. Je-li například `static int promenna;` globální proměnná, je viditelná pouze v daném souboru zdrojového textu (modulu). Z jiného modulu k ní nelze přistoupit. Obdobně může být `static` použito i u funkcí. Funkce je potom viditelná pouze v daném modulu. Je-li `static int promenna;` lokální proměnná funkce, potom její hodnota je uchovávána mezi jednotlivým voláním. Proměnná vzniká prvním zavoláním funkce a zaniká při ukončení programu. Tolik (stručně) k použití `static` v jazyce C. V jazyce C++ máme ale navíc dvě další možnosti jak použít klíčové slovo `static`. Můžeme ho použít u členské proměnné ve třídě, nebo u metody.

Klíčové slovo static a členská proměnná

Je-li klíčovým slovem `static` označena členská proměnná, jedná se o atribut, který všechny instance dané třídy sdílejí. Atribut existuje (lze s ním pracovat) i v případě, že neexistuje žádná instance třídy. Často se používá k počítání instancí dané třídy. Lze k ní přistupovat pomocí identifikátoru třídy, symbolu :: a názvu proměnné. Vše si ukážeme níže v příkladu.

Klíčové slovo static a členská metoda

Je-li klíčovým slovem `static` označena členská metoda, jedná se o metodu, která nemá jako svůj první implicitní parametr `this`. Není tedy těsně spojená s nějakou konkrétní instancí. V těle metody nelze pracovat s "nestatickými" atributy třídy a nelze volat "nestatické" metody třídy. Je to logické, protože není jasné pro který objekt je volána (není parametr `this`). Lze ji také zavolat, aniž by existovala nějaká instance dané třídy. Příklad:

```
#include<iostream>
using namespace std;

class Trida
{
    private:
        static int PocetInstanci = 0;
        int Atribut;
    public:
        Trida() : Atribut(0) { PocetInstanci++; }
        ~Trida() { PocetInstanci--; }
        static int dejPocetInstanci() { return PocetInstanci; }
        int operaceSNestatickymAtributem(int a)
             { int temp = Atribut; Atribut = a; return temp; }
};

int main(int, char**)
{
    Trida objekt1, objekt2, *objekt3 = new Trida();
    cout << Trida::dejPocetInstanci() << endl;
    cout << objekt2.operaceSNestatickymAtributem(10) << endl;
    cout << objekt1.operaceSNestatickymAtributem(10) << endl;
    cout << objekt3->operaceSNestatickymAtributem(10) << endl;
    cout << objekt3->operaceSNestatickymAtributem(5) << endl;
    delete objekt3;
    cout << Trida::dejPocetInstanci() << endl;
    return 0;
}
```

V "čistě" objektově orientovaných jazycích (mezi které určitě nepatří C++) se třída považuje za objekt. Je to objekt, který má svou identitu, stav a svému okolí poskytuje nějaké služby. Může se například jednat o službu "vytvoření své instance". (Vznikají zde zajímavé problémy. Je-li třída objekt, co je její třídou? Tedy co je třídou třídy? Je to metatřída, což je opět objekt. Co je třídou metatřídy? atd... S podobnými "žertíky" se člověk může setkat například ve Smalltalku.) V této souvislosti lze atributy dělit na instanční atributy (atributy objektu třídy - každý objekt má své) a třídní atributy (atributy, které má třída). Díky klíčovému slovu `static` můžeme považovat atributy a metody označené jako `static` za třídní atributy a metody, naopak atributy a metody neoznačené jako `static` můžeme považovat za instanční atributy a metody. Není to ale přesná analogie, protože v C++ NENÍ třída objektem. Pro představu, jak jsem to myslel, uvádím poslední příklad:

```
#include<iostream>
using namespace std;

class Trida
{
    private:
        static unsigned int PocetInstanci = 0; // "Třídní" proměnná
        int Atribut;                           // "InstanČní" proměnná
    public:
        Trida() : Atribut(0) {PocetInstanci++;}
        ~Trida() { PocetInstanci--; }
        static Trida *vytvorInstanci() { return new Trida(); }
        static dejPocetInstanci() { return PocetInstanci; }
        int vrat() const { return Atribut; }
        void nastav(unsigned int a) { Atribut = a; }
};

int main(int, char**)
{
    cout << "Pocet instanci:" << Trida::dejPocetInstanci() << endl;
    Trida objekt1;
    cout << "Pocet instanci:" << Trida::dejPocetInstanci() << endl;
    objekt1.nastav(3);
    cout << objekt1.vrat() << endl;
    Trida  *objekt2 = Trida::vytvorInstanci(),
       *objekt3 = Trida::vytvorInstanci();
    cout << objekt2->vrat() << objekt3->vrat() << endl;
    cout << "Pocet instanci:" << Trida::dejPocetInstanci() << endl;
    delete objekt2;
    delete objekt3;
    cout << "Pocet instanci:" << Trida::dejPocetInstanci() << endl;
    return 0;
}
```

Tímto bych seriál ukončil. Snažil jsem se podrobněji popsat jazyk C++ pro ty, kteří znají C. Chtěl jsem ukázat vymoženosti jazyka C++ oproti C. Snad se mi to alespoň trochu povedlo. Mezi vymoženosti C++ oproti C patří především principy OOP, přetěžování operátorů, parametrizace pomocí šablon a STL. Původně jsem ani nepočítal s tím, že seriál bude mít tak mnoho dílů. Už je skoro jako Esmeralda :-) Děkuji všem, kteří seriál dočetli až sem. Také děkuji všem, kteří mě v komentářích pod článkem upozornili na nějaké chyby a měli se mnou trpělivost. Chyb bylo opravdu dost.

Na úplný závěr ještě uvedu zdroje, ze kterých jsem čerpal. Jedná se spíše o typy pro Vás, pokud se o jazyk C++ více zajímáte. O některých jsem věděl

již dávno, na některé mě upozornili čtenáři v komentářích pod články.

- Cay S. Horstman: Vyšší škola objektově orientovaného návrhu v C++, Science 1997
- Stanislav Racek, Martin Kvoch: Třídy a objekty v C++, Kopp 1988
- Pavel Herout: Učebnice jazyka C, Kopp 1996
- Chip ročníky 1999 - 2001, články zaměřené na C++. Autoři jsou převážně pánové J. Franěk a M. Virius.
- Dokumentace k STL - http://www.sgi.com/tech/stl/stl_index_cat.html
- www.cplusplus.com
- Manuálové stránky OS LINUX
- Elektronická dokumentace a nápověda k překladači Borland C++ Builder®
- Elektronická dokumentace a nápověda k překladači GNU C/C++
- Elektronická dokumentace a nápověda k překladači Mingw