Dynamika objektů

Karel Richta a kol.

Katedra technických studií
Vysoká škola polytechnická Jihlava

Objektově-orientované programování, OOP
02/2020, Lekce 3

https://moodle.vspj.cz/course/view.php?id=200875

Inicializační seznam

```
class Trojuhelnik {
 double a, b, c;
public:
    Trojuhelnik(double stranaA, double stranaB, double stranaC);
};

Trojuhelnik::Trojuhelnik(double stranaA, double stranaB, double stranaC) :
a(stranaA), b(stranaB), c(stranaC) {}
```

```
// ekvivalentní
Trojuhelnik::Trojuhelnik(double stranaA, double stranaB, double stranaC) {
   a=stranaA; b=stranaB; c=stranaC;
}
```

Konstantní členská funkce – jen dotaz

- Zajistí, že objekt nebude měněn – jedná se o pouhý dotaz na data.
- Uvede se klíčové slovo **const** za seznam argumentů.

```
class Auto{
  int objem; string spz;
public:
  Auto(string s, int obj) { spz = s; objem = obj; }
  Auto(){}; //Implicitní Kurzor
  void zobrazAuto() const;
};
void Auto::zobrazAuto() const {
  cout << "Auto: spz " << spz << " objem "<< objem << endl;
}
int main() {
  Auto a1 = Auto("10-ABC-10",1500);
  Auto a2("55-AAA-55",1000);
  a1.zobrazAuto();
  a2.zobrazAuto();
  return 0;
}
```

Ukazatel this

- Každý objekt má ukazatel **this** – obsahuje adresu tohoto objektu – ukazuje na něj.
- Každá členská funkce jej může použít (skrytý argument všech metod).
- Použit při konfliktu členských dat a argumentu metody.

```
class Auto {
int objem; string spz;
public:
  Auto(string spz, int objem) {
     this->spz = spz; this->objem = objem; }
  int getObjem() const { return objem; }
  string getSpz();
  void test(Auto&);
  const Auto& vetsiObjem(const Auto& aut) const ;
};

string Auto::getSpz() { return this->spz; }
```

zde není nutné

Ukazatel this

- Každý objekt má ukazatel **this** – obsahuje adresu tohoto objektu – ukazuje na něj.
- Použit pro test přiřazení sebe sama.
- ***this** lze použít jako odkaz na objekt jako celek.
- Problém jak porovnat 2 objekty.

```
void Auto::test(Auto & aut) {
  if (&aut==this) cout << "Stejny objekt" << endl;
  else cout << "Jiny objekt" << endl;
}
const Auto& Auto::vetsiObjem( const Auto& aut) const {
  if (aut.objem > objem) return aut;
  else return *this;
}
```

```
Auto a1("HBA 21-07", 1500);
cout << a1.getSpz() << "; objem " <<a1.getObjem() << endl;
Auto a2 = Auto("HBB 21-07", 1000);
cout << "a1 a a2: "; a1.test(a2);
cout << "a1 a a1: "; a1.test(a1);
cout << "vetsi objem z aut a1 a a2: " << a1.vetsiObjem(a2).getObjem();
```

```
HBA 21-07; objem 1500
a1 a a2: Jiny objekt
a1 a a1: Stejny objekt
vetsi objem z aut a1 a a2: 1500
```

Organizace projektu v C ++

- Hlavičkový soubor – deklarace třídy, případně vložené funkce - přípona *trida*.h (header, někdy .hpp).
- Soubor *trida*.cpp se stejným jménem jako hlavička – definice třídy, implementace metod (zahr – stdafx.h).
- Soubor *projekt*.cpp s názvem projektu- tělo main a vlastní algoritmus řešení (zahr – stdafx.h).

```
//auto.h
#ifndef  __AUTO_H
#define  __AUTO_H
#include <iostream>
using namespace std;

class Auto {
private: int objem; string spz;
public:
  Auto(string s, int obj);
  string getSpz();
  int getObjem();
  Auto() { spz = „XXX-00-00"; }
  ~Auto() { cout << "\nrusim "<< spz;}
};
#endif
```

```
//auto.cpp
#include "stdafx.h"
#include "auto.h"
int Auto::getObjem() { return(this->objem); }
string Auto::getSpz() { return(this->spz); }
```

```
//projekt.cpp
#include "stdafx.h "
#include <iostream>
#include "auto.h"

int main() {
  Auto a1 ();
  std::cout<<"\n SPZ: "<<a1.getSpz()<<
  return 0;
}
```

stdafx.h

- používané v MS Visual Studio
- při první kompilaci programu se do souboru "stdafx.h" uloží zkompilovaná verze všech hlavičkových souborů programu
- při každé kompilaci programu kompilátor vybere zkompilovanou verzi hlavičkových souborů ze souboru "stdafx.h", místo aby znovu a znovu kompiloval tytéž hlavičkové soubory od začátku

- Výhody použití
  - Zkracuje dobu kompilace těch programů, které je nutné opakovaně kompilovat.
  - Snižuje zbytečné zpracování.

Konvence C++

- Funkce **main** vrací hodnotu **int** – pro testování, 0 – v pořádku. To platí jako konvence i pro řadu knihovních funkcí.
- Identifikátor funkce – začíná malý písmenem.
- Identifikátor pojmenovaných konstant – jen velká písmena MAX_ITERATION.
- Pokud chcete, aby Váš program šlo zveřejnit - použijte angličtinu.
- Funkce a proměnné – lowerCamelCase : getName(), setValueX(int x).
- Třídy – UpperCamelCase.
- Vlastní typy – UpperCamelCase.
- Globální proměnné pokud možno nepoužívat, případně vždy přístup přes ::, nejlepší varianta – singleton.
- Soukromé položky psát se suffixem _ nebo prefixem - m_var:

```
class SomeClass { private: int length_; int m_count; }
```

- Formální argumenty u vlastních typů – stejné jméno jako typ:

```
void setTopic(Topic* topic) // NOT: void setTopic(Topic* value)
                            // NOT: void setTopic(Topic* aTopic)
                            // NOT: void setTopic(Topic* t) void
connect(Database* database)  // NOT: void connect(Database* db)
                            // NOT: void connect (Database*
                                    oracleDB)
```

Konvence C++ (pokr.)

Pro funkce (metody):

- **get** – získání hodnoty členských dat : getName().
- **set** – nastavení hodnoty členských dat : setName (name).
- **find** – hledání : findElement().
- **compute** – výpočet: computeAveragePrice().
- **initialize** – inicializace objektu, konceptu.
- **is** – predikát - metoda vrací boolean: isVisible(), isTrue().
- Množné číslo na skupinu objektů: int values[].
- Počet objektů – prefix n: nLines.
- Pojmenované ukazatele – Line * line (Line * pLine).
- Vyhnout se členským datům **public**.
- **NULL** je používáno v C (makro), v C++ použít **0**, nebo **nullptr** (klíčové slovo, literál, definováno C++11).

The End