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

Objektově orientovaný styl

- Hlavní zásady:
  - Při rozkladu problému specifikujeme nové datové typy (datové abstrakce), tzn. specifikujeme atributy, které budou objekty charakterizovat a dále specifikujeme operace, které se budou s atributy těchto objektů provádět.
  - Pro realizaci datových abstrakcí použijeme objekty a třídy.
- Charakteristika objektu:
  - objekt se může nacházet v různých stavech daných hodnotami svých datových položek (atributů),
  - chování objektu je dáno metodami (operacemi), které jsou pro něj definovány třídou,
  - datové položky a metody objektu jsou dány typem objektu – třídou,
  - datové položky má každý objekt své vlastní,
  - metody jsou společné pro celou třídu a realizují určitý algoritmus.

Příklad

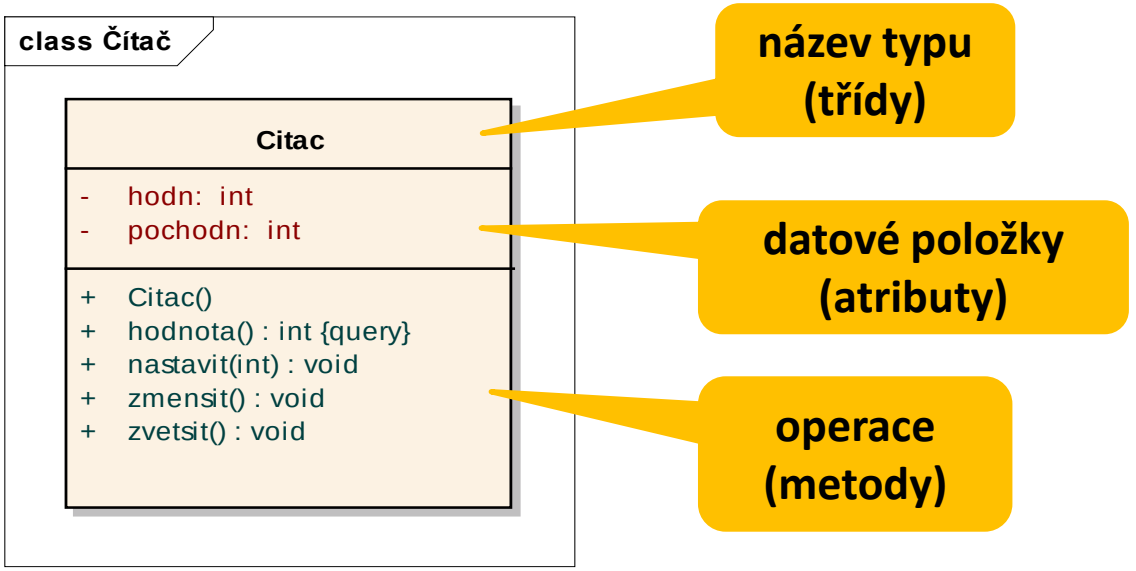
- Popelnice
  - konstruktor: vytvoř popelnici
  - příkazy: přidej odpad, vyprázdni se
  - dotazy: kolik je odpadu v popelnici?, je otevřeno víko?

Základy objektové orientace

- OO analýza a návrh – UML a metodiky.
- OO implementace – OO jazyky.
- Třída = uživatelem definovaný typ (abstrakce), který modeluje část reality.
- Třída = šablona pro vytváření objektů.
- Třída má vlastnosti a chování, nelze k nim přistupovat (měnit je za běhu).
- Objekt = reálný výskyt (instance) třídy, zhmotnění třídy, svět je tvořen objekty.
- Objekt lze vytvořit, používat, měnit jeho vlastnosti i odstranit.
- Velice zjednodušeně je třída typ a objekt je hodnota tohoto typu.

Příklad: Čítač objektově jako datový typ

- Čítač zavedeme jako datový typ s operacemi:
  - zvetsit, zmensit, nastavit a hodnota
- a s datovými položkami:
  - hodn a pocHodn
- Grafické vyjádření:



- Poznámka: hodnota položky pocHodn bude stanovena při vytvoření objektu typu Citac – můžeme vytvářet čítače s různými počátečními hodnotami.

Třída „Citac“

```
/* citac.h */

class Citac {
        int hodn;
        int pocHodn;
public:
        Citac(int ph) { pocHodn = ph; nastavit(); }
        void zvetsit() { hodn++; }
        void zmensit() { hodn--; }
        void nastavit() { hodn = pocHodn; }
        int hodnota() { return hodn; }
};
```

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

class Auto{
        string spz;
        int objem;
public:
        void setSpz(string s) { spz = s; } // deklarace s definicí
        void setObjem(int obj ) { objem = obj; }
        string getSpz(){ return spz; }
        inline int getObjem() { return objem; }
        // překladač vytvoří destruktor a bezparametrový konstruktor
};
```

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

class Auto{
private:
        string spz;
        int objem;
public:
        void setSpz(string s) { spz = s; } // deklarace s definicí
        void setObjem(int obj ) { objem = obj; }
        string getSpz(){ return spz; }
        inline int getObjem() { return objem; }
};
```

```
class Auto{
        string spz;
        int objem;
public:
        void setSpz(string s);   // deklarace bez definice, zakončena středníkem
        void setObjem(int obj );
        string getSpz();
        int getObjem();
};
void Auto::setSpz(string s) { spz = s; }  // definice, bez středníku
void Auto::setObjem(int obj ) { objem = obj; }
string Auto::getSpz(){ return spz; }
inline int Auto::getObjem() { return objem; }
```

```
class Auto{
        string spz;
        int objem;
public:
        void setSpz(string s) { spz = s; } // deklarace s definicí
        void setObjem(int obj ) { objem = obj; }
        string getSpz(){ return spz; }
        int getObjem(); // deklarace bez definice, zakončena středníkem
};
inline int Auto::getObjem() { return objem; }
```

Vytvořte třídu Trojúhelník

- trojúhelník je tvořen pomocí 3 stran, pro které by mělo jít:
  - nastavit délku (jedné strany i všech najednou)
  - zjistit délku každé jedné strany
- dále by objekt třídy Trojúhelník měl umožnit:
  - zjistit
    - obvod
    - obsah (Heronův vzorec)
    - zda se jedná o rovnostranný nebo rovnoramenný
    - zda se jedná opravdu o trojúhelník (pro délky jeho stran musí něco platit)
    - zda se jedná o pravoúhlý trojúhelník (kosinova věta)
  - metodou tisk() vypsat na obrazovku délky stran

```
class Auto{
        string spz;
        int objem;
public:
        void setSpz(string s) { spz = s;
        void setObjem(int obj ) { obje
        string getSpz(){ return spz; }
        int getObjem();
};
inline int Auto::getObjem() { return o
```

Všechny metody tříd mají jeden implicitní argument, který se neuvádí v deklaraci, ale protože je to metoda třídy, má ukazatel na objekt, se kterým má pracovat.

```
int main() {
        Auto a1, a2;
        a1.setSpz("6A6-10-10");
        a1.setObjem(1500);
        a2.setSpz("8DD-20-20");
        a2.setObjem(1000);
        cout << "objem: a2=" << a2.getObjem()
                << " a1="<<a1.getObjem()
                << endl;
        cout << "SPZ: a2=" << a2.getSpz()
                << " a1=" <<a1.getSpz()
                << endl;
        return 0;
}
```

```
objem: a2=1000 a1=1500
SPZ: a2=8DD-20-20 a1=6A6-10-10
```

Ověření práce s třídou Trojúhelník

- Vyzkoušejte všechny vytvořené metody

Vlastnosti Objektů

- Vlastnosti zajišťují unikátnost objektů:
  - Ovlivňují jakým způsobem se metody na objektu chovají.
  - Některé vlastnosti mohou být konstantní, jiné se mohou měnit.
  - Vlastnosti mohou být také objekty.
- Vlastnosti *objektů* dělíme na:
  - atributy - objekty, které pomáhají *objekt* popsat,
  - komponenty - objekty, které jsou součástí daného *objektu* (objekt se z nich skládá, např. auto *má* kola),
  - asociace - objekty o kterých daný *objekt* ví, ale nejsou jeho součástí (např. auto je v garáži, auto může vědět o garáži).

Stav Objektu

- Stavem *objektu* nazýváme kolekci všech jeho vlastností společně s jejich hodnotami.
- Stav *objektu* se změní pokud se změní aktuální hodnota(y) některé jeho vlastnosti(í).
- Mějme *objekt Auto* s následujícími vlastnostmi:
  - rok výroby,
  - barva.
- Co popisuje stav tohoto *objektu*?
- Může se změnit *barva*? Ano, auto můžeme nechat přebarvit.
- Může se změnit *rok výroby*? Ne, jedná se o konstantní vlastnost.

Schopnosti Objektů

- Schopnosti (*metody*) objektů umožňují provádět specifické akce.
- Metoda objektu musí být vyvolána nějakým objektem (včetně objektu, na kterém je metoda volána).
- Metody objektů dělíme na:
  - konstruktory - inicializují počáteční stav objektu
  - příkazy („set“ metody) - mění vlastnosti, tudíž stav objektu
  - dotazy („get“ metody) - poskytují odpověď na základě vlastností, stavu objektu
  - destruktory – ruší objekty

Datové typy C++

- Základní datové typy:
  - celočíselné: short, int, long
  - reálné: float, double, long double
  - znaky a řetězce: char (znak) a pole znaků char p[10]; (hlavně v C)
  - předdefinované řetězce – string (v C++)
  - prázdný typ: void
- Speciální, strukturované:
  - pole – int pole[10] – homogenní struktura
  - výčtový typ, enum barva{R,G,B};
  - struktura (struct) – heterogenní struktura (hlavně v C)
  - třída (class) – heterogenní struktura (v C++)

Opakování: Globální a lokální proměnné

- Proměnné jsou lokální a globální (deklarace, definice(inicializace)).
- Lokální proměnná má platnost v bloku.
- Globální proměnná má platnost ve všech funkcích a metodách (může ji zakrýt lokální deklarace).

```
int a = 0, b = 0, c = 1; /* globální proměnné */
void funkce(int a, int b) //formální argumenty {
//int a = -5; lokální proměnné - zde nelze předefinovat
   cout << "main: a = " << a << ", b = " << b << ", c = " << c << endl;
}
int main(void) {
  int c = 25;
  cout << "main: a = " << a << ", b = " << b << ", c = " << c << endl;
  funkce(100, 200);
  return 0;
}
```

```
main: a = 0, b = 0, c = 25
main: a = 100, b = 200, c = 1
```

Opakování: Globální a lokální proměnné

- V C++ lze pro určení globální proměnné použít operátor čtyřtečky :: .

```
int a = 0, b = 0, c = 1; /* globální proměnné */
void funkce(int a, int b) //formální argumenty {
//int a = -5; lokální proměnné - zde nelze předefinovat
   cout << "main: a = " << ::a << ", b = " << b << ", c = " << c << endl;
}
int main(void) {
  int c = 25;
  cout << "main: a = " << a << ", b = " << b << ", c = " << c << endl;
  funkce(100, 200);
  return 0;
}
```

```
main: a = 0, b = 0, c = 25
main: a = 0, b = 200, c = 1
```

cmath

- https://www.programiz.com/cpp-programming/library-function/cmath

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
	// computes 5 raised to the power 3
	cout << pow(5, 3);
	return 0;
}
// Output: 125
```

Základní pilíře OOP

- **Zapouzdření** (kód je pohromadě se zpracovávanými daty):
  - Skrývání implementace (nikdo nemá mít šanci zjistit, jak je program implementován).
  - Zvýšení bezpečnosti a robustnosti (nemožnost nekorektního použití).
  - Usnadnění budoucích modifikací.
- **Polymorfismus**:
  - Každá zpráva musí mít svého adresáta, nelze ji poslat „do prostoru“.
  - Objekt sám rozhodne, jak na zprávu zareaguje.
- **Skládání**:
  - Objekt může obsahovat jiné objekty.
- **Dědičnost**:
  - Speciální případ, při němž s objektem převezmu i jeho rozhraní.
  - Omezuje duplicity v kódu.
  - Nebezpečí špatného použití (narušuje zapouzdření).
  - Používání návrhových vzorů.
  - Proč bych měl vymýšlet něco, co už je vymyšlené, a je to vymyšlené dobře.

Členská data třídy

Datové položky třídy:

- Popisují datový typ položek, které u třídy definujeme.
- Standardní je jejich nepřístupnost mimo metody třídy => zapouzdření dat.

K určení přístupu ke členským datům slouží :

- **Private** (soukromé) – pouze veřejné členské metody daného objektu mohou přistoupit k položkám, přes identifikátor objektu a tečkovou notaci.
- **Public** (veřejné) – libovolný program může přistupovat k položkám, přes identifikátor objektu a tečkovou notaci.
- Doporučení: členská data vždy v části private => ukrývání dat.

```
class Auto{
private:
  string spz;
  int objem;
public:
  void setSpz(string spz);
  void setObjem(int obj );
};
```

```
int main() {
  Auto a1;
  a1.spz = "HBA-07-21"; // nelze
  a1.setSpz("ABC-20-10");
  a1.setObjem(100);
  // nelze
  // a1.objem = 100
  return 0;
}
```

Konstruktory

- Konstruktor můžeme vnímat jako metodu, která se jmenuje stejně jako třída, jejíž instanci vytváří a typ návratové hodnoty se neuvádí – je to nová instance třídy.
- Vytvoření instance třídy (objektu) je shodné s vytvořením proměnné uživatelského typu ------ **`Auto a1;`**
- Motivace tříd – činnost jako s datovým typem, ale:
  - **`a1 = {10,”10-BAC-20”}; // nelze`**
- Nutnost inicializační členské metody, která bude zavolána dříve než ostatní metody, tj. první = konstruktor.
- Konstruktor je automaticky volán při vytvoření objektu.
- Konstruktor – členská metoda = prototyp i definici.
- Konstruktor nesmí mít návratový typ – ani **void**.
- Konstruktor má stejné jméno jako třída.
- Konstruktor nelze vyvolat přes tečkovou notaci.

Příklad

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

class Auto{
private:
  int objem; string spz;
public:
  void initializeAuto(string s, int obj) {
        spz = s; objem = obj;
  }
  Auto(string s, int obj) {
        spz = s; objem = obj;
  }
  string getSpz(){ return spz; }
};

int main() {
  Auto a1 = Auto("10-ABC-10",1500);
                        //a1.initializeAuto("10-ABC-10",1500);
  Auto a2("00-YYY-00",1000);
  cout << a1.getSpz() << " " << a2.getSpz();
  return 0;
}
```

Konstruktor – členská metoda

- Nemusí být vložený, lepší je oddělit jeho deklaraci a definici.
- Zajišťuje konzistenci objektu.
- Může být přetížen – může existovat více konstruktorů pro stejnou třídu. Nemohou se lišit jménem (to je stejné), musí se ale lišit argumenty.
- Standardně použit pro inicializaci členských dat.

```
class Auto{
private:
  int objem; string spz;
public:
  string getSpz () { return spz; }
  Auto(const char * spz, int obj);
};
Auto::Auto(const char * spz, int obj){ strcpy(spz,spz);objem = obj; }
int main() {
  Auto a1 = Auto("10-ABC-10",1500); Auto a2("00-YYY-00",1000);
  cout << a1.getSpz() << " " << a2.getSpz();
  return 0;
}
```

Implicitní konstruktor (default)

- Implicitní konstruktor je vytvořen automaticky překladačem.
- Je použit pro inicializaci objektu, u kterého nebyla zadána žádná inicializační data.
- Je to konstruktor bez argumentů.
- Odpovídá vytvoření proměnné bez její inicilizace: `int i;`
- Obdobně: `Auto a1;`

```
class Auto {
private:
  int objem; string spz;
public:
  Auto(string s, int obj) { spz = s; objem = obj; }
  string getSpz() { return spz; }
  void setSpz(string s) { spz = s; }
  Auto() {};// Implicitni konstruktor
};
```

Přetížení konstruktorů

- Konstruktory lze přetěžovat, musí se ale poznat, který se má vyvolat.
- Pokud vytvořím vlastní variantu konstruktoru (explicitní) – překladač nevytvoří implicitní – je vhodné ho dotvořit vlastními prostředky – doporučení vždy vytvořit.
- Implicitní konstruktor lze řešit jako variantu explicitního konstruktoru s implicitními argumenty.

```
class Auto {
private:
  int objem; string spz;
public:
  Auto(string s, int obj) { spz = s; objem = obj; }
  string getSpz() { return spz; }
  void setSpz(string s) { spz = s; }
  Auto() { spz = "00-000-00"; objem = 0; };  // Implicitni konstruktor
//Auto(string s = "00-000-00", int obj = 0) { spz = s; objem = obj; }
};
```

Možné způsoby vzniku objektu (1/2)

`Obj o;`
`Obj o{};`
`Obj* o = new Obj;`
`Obj* o = new Obj();`
`Obj* o = new Obj{};`
	V těchto případech se zavolá výchozí konstruktor.

`Obj o = 42;`
`Obj o(42);`
`Obj o{42};`
`Obj* o = new Obj(42);`
`Obj* o = new Obj{42};`
	V těchto případech se zavolá konstruktor, který přijímá parametr typu `int`.

`Obj o[42];`
`Obj o[42]{};`
`Obj* o = new Obj[42];`
`Obj* o = new Obj[42]();`
`Obj* o = new Obj[42]{};`
	V těchto případech se 42-krát zavolá výchozí konstruktor.

`Obj o();`	Zrada (známá jako *most vexing parse*): toto není vznik objektu, ale deklarace funkce.

Možné způsoby vzniku objektu (2/2)

`void receivesObj(Obj o); ...`
`receivesObj({});  // výchozí`
`receivesObj(42);  // přijímající int`
	Vznik objektu až při předávání hodnoty do funkce

`Obj returnsObj() { ...`
`  return {};  // výchozí`
`  return 42;  // přijímající int`
	Vznik objektu při vracení hodnoty z funkce

`... Obj() ...;  // výchozí`
`... Obj{} ...;  // výchozí`
`... Obj(42) ...;  // přijímající int`
`... Obj{42} ...;  // přijímající int`
	Vznik dočasného *(temporary)* objektu – objekt nemá název

Rozšíření třídy Trojúhelník

- Vytvořte konstruktory pro trojúhelník:
  - rovnostranný (1 parametr)
  - rovnoramenný (2 parametry)
  - libovolný (3 parametry)

- Vyzkoušejte

Destruktor *(destructor)*

- Vždy, když zanikne instance nějaké třídy, je skrytě (implicitně) zavolána metoda zvaná **destruktor**.
- Destruktor třídy T je její metoda pojmenovaná ~T.
- Rozdíly oproti konstruktoru:
  - Nemá parametry.
  - Nemá inicializační seznam.
  - V dané třídě může být pouze jeden.

```
class Person {
    ~Person() {
        cout << "destruktor\n";
    }
};
```

Význam destruktoru

- Provádí úklid; uvolňuje **prostředky**, za které má daná instance **odpovědnost**.
  - Prostředky mohou být: paměť, soubory, připojení k databázím, a další.

```
class User {
public:
    User() {
        *data = new UserData;       // získej paměť
        systemLogIn(*this);         // přihlas se
    }
    ~User() {
        systemLogOut(*this);        // odhlas se
        delete data;                // uvolni paměť
    }
private:
    UserData* data;
};
```

Příklady 1 až 3

Statické členy třídy - statické proměnné

- Statické členy třídy patří třídě, nikoliv objektu (instanci), tzn. statické proměnné můžeme použít aniž bychom vytvořili objekt.
- Statické proměnné inicializujeme v samostatné definici mimo třídu.

```
class Auto{
public:
        static int pocet_aut; // nelze přiřadit hodnotu
        inline int getPocetAut() { return pocet_aut; }
};

int Auto::pocet_aut=5;
```

https://physics.ujep.cz/~mmaly/vyuka/progC/material_EX6.pdf

Statické členy třídy - statické proměnné

```
int main() {
        Auto::pocet_aut=10;
        cout << "hodnota staticke promenne: " << Auto::pocet_aut << endl;
        Auto a1;
        Auto::pocet_aut=20;
        cout << "hodnota staticke promenne: " << a1.pocet_aut << endl;
        a1.pocet_aut++;
        cout << "hodnota staticke promenne: " << a1.getPocetAut() << endl;
        return 0;
}
```

```
hodnota staticke promenne: 10
hodnota staticke promenne: 20
hodnota staticke promenne: 21
```

Rozšíření třídy Trojúhelník

- Vytvořte veřejnou statickou proměnnou třídy Trojúhelník pro počítání počtu instancí třídy Trojúhelník (je třeba upravit konstruktory – zvýšení o 1, ale i destruktor – snížení o 1)
- Vyzkoušejte

Dynamické proměnné

- Kromě lokálních a globálních objektů (statických proměnných) existuje ještě jedna možnost, jak objekty dynamicky vytvářet – na tzv. haldě (heap).
- Objekty na haldě vytváří v C++ explicitně programátor – pomocí operátoru **new**.
- Ruší je naopak pomocí operátoru **delete**.
- Operátory **new** a **delete** využívají konstruktory a destruktory – ty probereme za chvíli.
- Příklad:

```
int main () {
    int *px = new int;
    *px = 7;
    delete px;
    int *pp = new int[10];
    for (int i = 0; i < 10; i++) pp[i] = i;
    delete [] pp;
    return 0;
}
```

Rozšíření třídy Trojúhelník

- Vyzkoušejte
  - jeden objekt vytvořte pomocí new a pak jej zrušte pomocí delete

```
Trojuhelnik* t2 = new Trojuhelnik(50,10,1);
cout << "t2.a = " << t2->getA() << endl;
delete t2;
```

Statické členy třídy – konstantní statické proměnné

- Statické proměnné můžeme zároveň definovat jako konstantní tam kde je to vhodné.

```
class Graphic
{
  public:
    // Nasledujici konstantni staticka
    // promenna bude pouzita pro cislo
    // modre barvy
    static const int COLOR_BLUE = 3;

};

int main()
{
  Graphic g1;
  cout << "Modra barva ma cislo: "
       << Graphic::COLOR_BLUE << endl;
  return 0;
}
```

https://physics.ujep.cz/~mmaly/vyuka/progC/material_EX6.pdf

Statické členy třídy - statické metody

- Klíčové slovo static uvádíme pouze v deklaraci ve třídě, v definici mimo třídu již ne.
- Statické metody se chovají podobně jako nečlenské funkce, tj. můžeme je volat aniž bychom vytvořili objekt.
- Statické metody mohou operovat pouze nad statickými daty dané třídy.

```
class Auto{
public:
        static int pocet_aut; // nelze přiřadit hodnotu
        static int getPocetAut() { return pocet_aut; }
};

int Auto::pocet_aut=5;
```

https://physics.ujep.cz/~mmaly/vyuka/progC/material_EX6.pdf

Statické členy třídy - statické metody

```
int main() {
        cout << "hodnota staticke promenne: " << Auto::getPocetAut() << endl;
        Auto a1;
        Auto::pocet_aut=20;
        cout << "hodnota staticke promenne: " << a1.pocet_aut << endl;
        a1.pocet_aut++;
        cout << "hodnota staticke promenne: " << a1.getPocetAut() << endl;
        return 0;
}
```

```
hodnota staticke promenne: 5
hodnota staticke promenne: 20
hodnota staticke promenne: 21
```

Zánik objektu

- Kdy objekt zaniká?
  - Lokální proměnná: když program opustí blok ({}), ve kterém objekt vznikl.
  - Datová položka třídy T: když zaniká instance třídy T, hned po provedení destruktoru ~T.
  - Objekt, který vznikl příkazem `new`: když zavoláme příkaz `delete` na ukazatel, který na objekt ukazuje.
  - Globální proměnná: když program opustí funkci `main`.
  - Dočasný objekt: po provedení řádku.
- Když zaniká více objektů najednou, tak zanikají přesně **v opačném pořadí**, než v jakém vznikaly.

Příklady: pořadí vzniku a zániku (1/3)

```
class Obj {
public:
    Obj(int num) : mNum(num) {cout << mNum << ' '; }
    ~Obj() {cout << '~' << mNum << ' '; }
private:
    int mNum;
};

int main() {
    Obj o1 = 1;
    Obj o2(2);
    Obj o3{3};
}
```

```
1 2 3 ~3 ~2 ~1
```

Příklady: pořadí vzniku a zániku (2/3)

```
class Obj {
public:
    Obj(int num) : mNum(num) {cout << mNum << ' '; }
    ~Obj() {cout << '~' << mNum << ' '; }
private:
    int mNum;
};

int main() {
    Obj o0 = 0;

    for (int i = 1; i <= 3; i++) {
        Obj o = i;
    }
}
```

```
0 1 ~1 2 ~2 3 ~3 ~0
```

Příklady: pořadí vzniku a zániku (3/3)

```
class Obj {
public:
    Obj(int num) : mNum(num) {cout << mNum << ' '; }
    ~Obj() {cout << '~' << mNum << ' '; }
private:
    int mNum;
};

void foo(Obj o) {
    cout << "foo ";
    Obj o2 = 2;
}

int main() {
    foo(1);
}
```

```
1 foo 2 ~2 ~1
```

The End