2. Třídy a objekty v C/C++

Ing. Peter Janků, Ph.D.
Ing. Michal Bližňák, Ph.D.

Ústav informatiky a umělé inteligence
Fakulta aplikované informatiky
UTB Zlín

Programování v jazyce C++, Zlín, 13. října 2021

Objektově orientované programování

Objektově orientované programování

- **O**bjektově **o**rientované **p**rogramování
  - **Zapouzdřuje** data (atributy) a funkce (operace nad daty) do jednoho logického celku.
  - Umožňuje **přirozeně modelovat** struktury **dat**, jejich **závislostí** a **operací** nad nimi.
  - Umožňuje implementovat **znovupoužitelné** bloky programu.
  - **Skrývá** vybrané implementační detaily.
- Základní termíny
  - **Třída** představuje základní jednotku OOP.
  - **Objekty** jsou **instanciované** třídy.

OOP vlastnosti

Základní předpoklady:

- Zapouzdření
  - znamená seskupení souvisejících dat a funkci do jednoho celku a zároveň řízení přístupu k těmto funkcím z jiných objektů

OOP vlastnosti

Základní předpoklady:

- Zapouzdření
  - znamená seskupení souvisejících dat a funkci do jednoho celku a zároveň řízení přístupu k těmto funkcím z jiných objektů
- Abstrakce
  - zobecnění konstrukce tříd a objektů

OOP vlastnosti

Základní předpoklady:

- Zapouzdření
  - znamená seskupení souvisejících dat a funkci do jednoho celku a zároveň řízení přístupu k těmto funkcím z jiných objektů
- Abstrakce
  - zobecnění konstrukce tříd a objektů
- Dědičnost
  - možnost přenášet vlastnosti a implementace na potomky a vytvářet tak třídy s obecnými vlastnostmi

OOP vlastnosti

Základní předpoklady:

- Zapouzdření
  - znamená seskupení souvisejících dat a funkci do jednoho celku a zároveň řízení přístupu k těmto funkcím z jiných objektů
- Abstrakce
  - zobecnění konstrukce tříd a objektů
- Dědičnost
  - možnost přenášet vlastnosti a implementace na potomky a vytvářet tak třídy s obecnými vlastnostmi
- Polymorfizmus
  - možnost využívání stejného přístupu k vlastnostem a funkcím objektu s různou implementací

OOP vlastnosti

Základní předpoklady:

- Zapouzdření
  - znamená seskupení souvisejících dat a funkci do jednoho celku a zároveň řízení přístupu k těmto funkcím z jiných objektů
- Abstrakce
  - zobecnění konstrukce tříd a objektů
- Dědičnost
  - možnost přenášet vlastnosti a implementace na potomky a vytvářet tak třídy s obecnými vlastnostmi
- Polymorfizmus
  - možnost využívání stejného přístupu k vlastnostem a funkcím objektu s různou implementací
- Kompozice
  - konstrukce složitější a komplexnějších objektů na základě předem vytvořených jednodušších.

Třídy jazyka C++

- Třídy
  - Modelují objekty s atributy a specifickými operacemi
  - Definice pomocí klíčových slov `class`, `struct` nebo `union`

Listing 1: Příklady třídy jazyka C++

```
1  class Time {
2  public:
3      void setSeconds(int s);
4      void setMinutes(int m);
5      void setHours(int h);
6      const char * getTime();
7
8  protected:
9      int m_seconds;
10     int m_minutes;
11     int m_hours;
12 };
```

Třídy jazyka C++

- `class`
  - Začátek **definice/deklarace třídy**
  - Tělo třídy uzavřeno ve složených závorkách {}
  - Definice/deklarace třídy ukončena středníkem ;
- `setSeconds`, `setMinutes`, `setHours`
  - Členské **funkce** (operace)
- `m_seconds`, `m_minutes`, `m_hours`
  - Členské **data** (atributy)
  - "Maďarská notace"
- `public`, `protected`
  - Specifikace přístupu/viditelnosti členů třídy

Členské metody a atributy tříd

Členská data (atributy) třídy

- Proměnná libovolného typu zapouzdřená jmenným prostorem třídy.
- Hodnoty, ukazatele i reference.
- Konstanty i modifikovatelné proměnné.
- Libovolná notace, doporučená "Maďarská" - `m_iName`.
  - `m_` - člen třídy (**m**ember)
  - `i` - datový typ (**i**nteger)
  - `Name` - vlastní jméno atributu

Listing 2: Data (atributy) třídy

```
1 class Data {
2     int variable;
3     const char * message;
4     float& pi;
5 };
```

Členské funkce (operace) třídy

- Funkce libovolného typu zapouzdřené jmenným prostorem třídy.
- Modifikace obsahu třídy.
- Čtení obsahu třídy.
- Konstrukce dat.

Listing 3: Funkce (operace) třídy

```
1 class Functions {
2     void sayHello();
3     void setHelloText(const char * msg);
4     const char * getHelloText() const;
5 };
```

Oddělení deklarace a definice

- Deklarace a definice členů třídy může být
  - **v místě/vložená/sloučená** (inplace/inline)
  - **oddělená**
- Způsob definice/deklarace **ovlivňuje překlad** zdrojového kódu.
  - Vložené definice mohou být vkládány do kódu **podobně jako makra**
- Oddělení deklarace a definice pomocí **modulů** a jejich **hlaviček**.
  - Definice rozhraní třídy (interface)
  - Implementace znovupoužitelné knihovny

Příklad sloučené deklarace a definice

Listing 4: Sloučená deklarace a definice

```
1  class Time {
2  public:
3      void setSeconds(int s) {m_seconds = s;}
4      void setMinutes(int m) {m_minutes = m;}
5      void setHours(int h) {m_hours = h;}
6
7  protected:
8      int m_seconds;
9      int m_minutes;
10     int m_hours;
11 };
```

Oddělená deklarace - time.h

Listing 5: Oddělená deklarace

```
1  class Time {
2  public:
3      void setSeconds(int s);
4      void setMinutes(int m);
5      void setHours(int h);
6
7  protected:
8      int m_seconds;
9      int m_minutes;
10     int m_hours;
11 };
```

Oddělená definice - time.cpp

Listing 6: Oddělená definice

```
1  #include "time.h"
2
3  void Time::setSeconds(int s) {
4      m_seconds = s;
5  }
6
7  void Time::setMinutes(int m) {
8      m_minutes = m;
9  }
10
11 void Time::setHours(int h) {
12     m_hours = h;
13 }
```

Přístup na členy třídy

- Přístup na členy třídy pomocí operátorů
  - **Tečka** .
    - **Hodnotový** přístup (globální paměť, zásobník)
  - **Šipka** ->
    - Přístup přes **ukazatele** (globální paměť, zásobník, halda)

Přístup na členy třídy

Listing 7: Přístup na členy třídy

```
1 Time time;
2 time.setHours(12);
3 time.setMinutes(30);
4
5 Time * pTime = &time;
6 pTime->setSeconds(59);
```

Konstantní funkce

- Funkce, které **nemohou měnit obsah rodičovského objektu** (instance třídy).
- **Omezení přístupu** k určitým členům.
- Lze volat také nad **konstantním objektem**
  - Striktní přístup pouze pro čtení
  - Nekonstantní funkci nad konstantním objektem nezavoláte...
- Třídy by měly být deklarovány jako "konstantně korektní" (const-correctness)
  - Použití zejména u konstantních kopírovacích konstruktorů[1]
- Syntaxe:

```
<type> <name> (<args>) const;
```

[1] Bude upřesněno později.

Definice konstantní funkce

Listing 8: Definice konstantní funkce

```
1  class ConstData {
2  public:
3      const char * getMessage() const {
4          return m_message;
5      }
6      void setMessage(const char * message) {
7          m_message = message;
8      }
9
10 protected:
11     const char * m_message = "Hello World!";
12 };
```

Volání konstantní funkce

Listing 9: Volání konstantní funkce

```
1 const ConstData d1;
2 d1.getMessage();          /* OK */
3 d1.setMessage("Bye!");    /* ERROR */
4
5 ConstData d2;
6 d2.setMessage("Bye!");    /* OK */
7 d2.getMessage();          /* OK */
```

Vložené (inline) funkce

- Funkce **dekorované** klíčovým slovem `inline`.
- **Doporučení** pro překladač.
- Funkce bude v místě volání vložena podobně jako makro a nebude realizován skok do podprogramu.
- **Optimalizace** běhu aplikace na **rychlost**.
- Při častém/opětovném použití může **zvýšit spotřebu operační paměti**.
  - Typicky u funkcí s velmi krátkou implementací
  - `get` a `set` funkce
  - Jednoduché výpočetní výrazy
- Příklad:

```
inline void setValue(int v) {m_value = v;}
```

Statické a instanční členy tříd

- Funkce a data dekorovaná klíčovým slovem `static`.
- Vždy pouze u oddělené deklarace a definice.
- Člen je přístupný bez nutnosti instanciace třídy pouze přes jmenný prostor třídy.
- Použití u sdílených členů/objektů tříd
  - Tzv. **singletony** - pouze jedna instance třídy pro celou aplikaci

Deklarace a definice statických členů

Listing 10: Deklarace statických členů

```
1 class StaticMembers {
2 public:
3     static void sayHello();
4     static const char * m_message;
5 };
```

Listing 11: Definice statických členů

```
1 const char * StaticMembers::m_message = "Hello!";
2
3 void StaticMembers::sayHello()
4 {
5     std::cout << StaticMembers::m_message << std::endl;
6 }
```

Volání statických členů

Listing 12: Volání statických členů

```
1    StaticMembers::sayHello();
2    const char * message = StaticMembers::m_message;
```

Viditelnost členů tříd

Viditelnost členů tříd

- Jazyk C++ rozlišuje tři typy viditelnosti členů tříd
  - `public`
  - `protected`
  - `private`
- Není-li specifikováno explicitně, je použit mód `private`
- `public`
  - Členy viditelné jak v rodičovské a odvozené[2] třídě, tak i vně.
- `protected`
  - Členy viditelné pouze pro vlastní a odvozené třídy.
- `private`
  - Členy viditelné pouze pro vlastní třídu.

[2] Bude upřesněno v další prezentaci.

Konstruktor a destruktor třídy

Konstruktor

- **Členská funkce** volaná při **vytváření instance (objektu)** třídy.
- Stejné jméno jako jméno třídy.
- Význam pro **inicializaci** objektu
  - Nastavení výchozí hodnoty jednotlivých členských atributů třídy[3]
  - Provedení specifických inicializačních kroků
- Konstruktor může být **s parametry, nebo bezparametrický**.
- Konstruktor **nespecifikuje návratový typ**.

[3] Norma C++11 umožňuje inicializovat členské atributy přímo v deklaraci třídy - není potřeba implementovat v konstruktoru.

Příklad použití konstruktoru

Listing 13: Bezparametrický konstruktor

```
1  class Data {
2  public:
3      Data() {
4          variable = 0;
5          message = NULL;
6      }
7
8  protected:
9      int variable;
10     const char * message;
11 };
```

Příklad použití konstruktoru

Listing 14: Konstruktor s parametry

```
1  class Data {
2  public:
3      Data() {;}
4      Data(int v, const char * m)
5          : variable(v), message(m) {;}
6
7  protected:
8      int variable = 0;                    /* C++11 */
9      const char * message = NULL;         /* C++11 */
10 };
```

Destruktor

- **Členská funkce** volaná při **uvolňování instance (objektu)** třídy.
- Stejné jméno jako jméno třídy předznamenané znakem ~
- Význam pro **deinicializaci** objektu
  - Uvolnění prostředků alokovaných při vytváření instance třídy nebo během jejího životního cyklu
  - Provedení specifických deinicializačních kroků
- Destruktor je **vždy bezparametrický**.
- Destruktor **nespecifikuje návratový typ**.

Příklad použití destruktoru

Listing 15: Definice třídy

```
1  class Buffer {
2  public:
3      Buffer(size_t size) {
4          std::cout << "Alocating " << size << " bytes."
               << std::endl;
5          m_buffer = new unsigned char[size];
6      }
7      ~Buffer() {
8          std::cout << "Releasing allocated buffer." <<
               std::endl;
9          delete [] m_buffer;
10     }
11 protected:
12     unsigned char * m_buffer = NULL;
13 };
```

Příklad použití destruktoru

Listing 16: Vytvoření instance třídy

```
1 int main(int argc, char** argv)
2 {
3     Buffer buffer(10);
4
5     return 0;
6 }
```

Listing 17: Výstup z programu

```
1 Alocating 10 bytes.
2 Releasing allocated buffer.
```

Děkuji za pozornost

A to je pro dnešek vše. Nastává čas pro vaše dotazy...