

12. Úvod do C++ (třídy a objekty)

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Třídy a objekty

Třídy

Zapouzdření

Objekty

Konstruktor

Vazba

Část I

Třídy a objekty

I. Třídy a objekty

Třídy

Zapouzdření

Objekty

Konstruktor

Vazba

Co je to třída?

- Třída je abstrakcí entity reálného světa.
- Příklad: třída Automobil:
 - všechny automobily mají nějaké společné vlastnosti
 - obsah motoru,
 - barvu,
 - ...
 - všechny automobily mají nějaké společné rozhraní (interface)
 - lze je nastartovat,
 - lze je rozjet nebo zastavit,
 - ...
- Objekt je tzv. instance třídy.
- Příklady instance třídy Automobil:
 - Škoda Octavia,
 - VW Passat,
 - ...

Pohled programátora

- Třída je popisem datového typu:
 - jméno,
 - data – členské proměnné (položky, atributy),
 - interface – členské funkce (metody).
- Třídy jsou vyvíjeny programátory a jsou kompilovány do spustitelného programu.
- V C++ nelze vytvářet nové třídy za běhu.
- Objekty (instance tříd) jsou proměnné:
 - každý objekt má třídu,
 - v průběhu běhu programu jsou objekty vytvářeny a rušeny,
 - obvykle vytváříme více objektů/instancí stejné třídy,
 - stejně jako jiné datové typy, C++ povoluje staticky alokované objekty, dynamicky alokované objekty, pole objektů, ...

Deklarace třídy

```
class T
{
    typ a; // deklarace položky (clenske promenne)
    typ f ( ... ); // deklarace metody (clenske funkce)
    T ( ... ); // deklarace konstrukturu
    ~T ( void ); // deklarace destrukturu
};
```

- Definice konstrukturu:

```
T::T ( ... ) { ... }
```

- Definice destrukturu

```
T::~~T ( void ) { ... }
```

- Definice metody

```
typ T::f ( ... ) { ... }
```

Inline metody

```
class T
{
    typ f ( ... ) { ... }
    T ( ... ) { ... }
    ~T (void) { ... }
};
```

- Inline metody jsou podobné inline funkcím
 - možná rychlejší, s vyššími paměťovými nároky
- Kompilátor se může rozhodnout nezkompilovat metodu jako inline
 - vypnuté optimalizace, příliš dlouhý kód, ...
- Inline metody snižují srozumitelnost deklarácí
- Všimněte si, že
 - deklarace metod jsou zakončeny středníkem,
 - definice metod středníkem zakončeny nejsou.

I. Třídy a objekty

Třídy

Zapouzdření

Objekty

Konstruktor

Vazba

Co je to zapouzdření?

- Pro úplnou ochranu stavových atributů objektu je potřeba zabránit jejich modifikaci jinak než přes příslušné metody.
- Atributy a metody třídy jsou vždy přístupné z metod definovaných v této třídě, z jiných metod a funkcí však již přístupné být nemusí.
- Zapouzdření je zároveň prostředek, jak vytvořit a udržovat kontrolovatelné a vysokoúrovňové veřejné rozhraní třídy. Je pak možné:
 - nezávisle upravovat implementaci uvnitř třídy (např. zvolit efektivnější algoritmus, jinou reprezentaci dat, ...), kompletně nahradit třídu bez rozbití zbytku programu (pokud nová třída dodrží veřejné rozhraní),
 - opravit chyby uvnitř třídy bez rozbití zbytku programu,
 - pracovat na vývoji tříd nezávisle (např. různými programátory zároveň).

Řízení přístupu

- Přístup ke členům třídy (metody / položky) je řízen pomocí **modifikátorů viditelnosti**:
 - `public` – jsou přístupné komukoli,
 - `protected` – jsou přístupné v třídě samé a v jejích podtřídách,
 - `private` – jsou přístupná jen v třídě samé.

```
class T
{
    // metody/položky s implicitním přístupem
    // zde private)
public:
    // metody/položky přístupne komukoli
private:
    // metody/položky přístupne jen ve tride
};
```

Klíčová slova `class` a `struct`

- Obě klíčová slova mohou být použita k deklaraci třídy
- Jediný rozdíl je v implicitní viditelnosti:

class: `private`

struct: `public`

```
class T {  
    // private je implicitni  
    int a;  
public:  
    void f ( );  
};
```

```
class T {  
public:  
    void f ( );  
private:  
    int a;  
};
```

```
struct T {  
private:  
    int a;  
public:  
    void f ( );  
};
```

```
struct T {  
    // public je implicitni  
    void f ( );  
private:  
    int a;  
};
```

I. Třídy a objekty

Třídy

Zapouzdření

Objekty

Konstruktor

Vazba

Vytváření objektů

- Objekt je proměnná (instance třídy).
- Při vytvoření objektu je zavolán konstruktor.
- Konstruktory mohou být přetíženy.
- Konstruktor nelze volat explicitně na existující objekt.
- Destruktor je volán automaticky, když je objekt rušen.

Příklad staticky alokovaných objektů:

```
int foo ( void ) {  
    T x (a, b); // konstruktor s parametry (a, b)  
    T y;       // konstruktor bez parametru  
    ...  
}
```

Přístup k metodám a položkám

- Metody jsou volány prostřednictvím tečkové notace.
- Metody mohou být přetíženy. Platí pravidla pro přetěžování funkcí (nejlepší shoda skutečných a formálních parametrů).
- Přístup k položkám (členským proměnným) je tečkovou notací (jako struct).

```
class T
{
    public:
        void foo ( ) { ... } // metoda
        int bar;           // položka
};
...
T x;
x.foo ();
x.bar = 10;
```

Dynamická alokace

- Objekty mohou být alokovány dynamicky – užitím operátoru `new`
Operátor volá odpovídající konstruktor.
- Dynamicky alokované objekty je rušeny použitím `delete`
Operátor volá destruktorku.
- C alokace (`malloc/free`) nemůže být použita.
Konstruktor ani destruktorka nejsou volány.

```
struct T
{
    int a;
    T (int x);
    ~T ();
    void f (int x);
};
```

```
T *p = new T (20);
p->a = 10;
p->f (15);
delete p;
```


Klíčové slovo `this`

- Lokální deklarace mohou být v konfliktu se jmény položek. V tomto případě jsou aplikována pravidla rozsahu platnosti jmen.
- Přístup k položkám lze zařídit pomocí klíčového slova `this` nebo plně kvalifikovaným jménem.
- Lepší je vyhnout se konfliktu jmen (např. prefixem).

```
struct T {  
    int a;  
    void f (int a);  
};  
void T::f (int a) { // konflikt se jménem položky a  
    T::a = a; // T::a - plně kvalifikované jméno položky  
    // a - jméno parametru  
    this->a = 10; // this - ukazatel na instanci typu T*  
}
```

Konstantní objekty a konstantní metody

- Metody mohou modifikovat položky.
- Objekt může být označen jako konstantní (const) – jeho položky pak nemohou být modifikovány.
- Metody mohou být označeny jako konstantní (const):
 - těla konstantních metod nemohou modifikovat položky
 - konstantní metody mohou být volány na jakémkoli objektu.
- Nekonstantní metody nelze na konstantním objektu volat.

kompilátor to kontroluje

Konstantní objekty a konstantní metody

```
class T
{
    int a;
public:
    T (int x) {a = x;}
    void print () {cout << a;}
    int get_a () const {return a;} // const metoda
};
const T x (1); // const objekt
...
cout << x . get_a (); // ok - konstantni metoda na
    konstantnim objektu
x.print (); // chyba - nekonstantni metoda na
    konstantnim objektu
```

I. Třídy a objekty

Třídy

Zapouzdření

Objekty

Konstruktor

Vazba

Volání konstruktoru

- Konstruktore je automaticky volán při vytvoření objektu.
- Po vytvoření objektu už další volání konstruktoru není dovoleno.
- Explicitní volání konstruktoru vytvoří nový nepojmenovaný dočasný objekt – může být použit pro výpočet (např. předání parametrů, lokální kopii, ...) a je pak zrušen.

```
class T
{
    int p, q;
public:
    T () {p=0; q=0;}
    T (int fp) {p=fp; q=0;}
};
T a, b (1);
a.T (1); // chyba, dalsi volani konstruktoru neni dovoleno
a = T (20); // je vytvoren novy docasny objekt, ten je
            zkopirovan do a, potom je docasny objekt zrusen
```

Implicitní konstruktor

- Konstruktor, který lze volat bez parametru, tj.:
 - nemá žádný formální parametr, nebo
 - jeho všechny formální parametry mají implicitní hodnoty.
- Implicitní konstruktor se použije při vytvoření nového objektu, nejsou-li uvedeny parametry.
- Když ve třídě není žádný konstruktor, systém automaticky generuje implicitně (prázdný) konstruktor.

```
class T
{
public:
    T (int a = 0) { ... } // implicitni konstruktor
};

// priklady pouziti
T a;
T b[10]; // implicitni konstruktor volan 10 krat
T *p = new T;
```

Statické (třídní) a instanční (členské) proměnné

- Instanční (členské) proměnné:
 - deklarované bez speciálního klíčového slova,
 - každá instance má své vlastní instanční proměnné.
- Statické (třídní) proměnné:
 - deklarované klíčovým slovem **static**, sdíleny instancemi třídy,
 - statické proměnné musí mít rezervovanou paměť – je požadována definice mimo deklaraci třídy,
 - přístup ke statickým proměnným se řídí standardními pravidly zapouzdření (public, protected, private).

```
class T {  
    int a; // instanční členská proměnná  
    static int cnt; // statická členská proměnná  
public:  
    T (int x) {a = x; cnt++;}  
    ~T () {cnt--;}  
};  
int T::cnt = 0; // definice (rezervace pameti)
```

Statické (třídní) a instanční metody

- Instanční metody:
 - mohou být volány na existující objekt,
 - mají přístup k instančním proměnným a mohou volat jiné instanční metodám, a to jak jak přímo v instanci, tak přes ukazatel `this`,
 - mají přístup k třídním proměnným a metodám.
- Statické (třídní) metody:
 - jsou podobné běžným funkcím,
 - nemají žádnou implicitní instanci, proto ani implicitní členské proměnné, ani ukazatel `this` nejsou k dispozici.
 - mají přístup k třídním proměnným a mohou volat jiné třídní metody.

```
class T {  
    static int cnt;  
public:  
    static int getCount () {return cnt;}  
};  
cout << T::getCount (); // volání statické metody
```


I. Třídy a objekty

Třídy

Zapouzdření

Objekty

Konstruktor

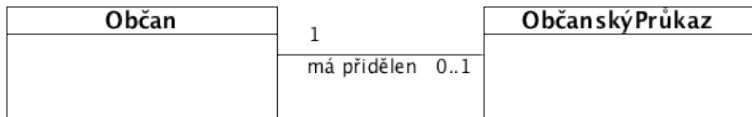
Vazba

Vazba

- V předchozí části přednášky jsme uvažovali jednoduché objekty, bez struktury, která by umožňovala vzájemnou **vazbu**.
Tento přístup je prakticky nepoužitelný – neumožňuje vystavět objektový model.
- Vazba¹ umožňuje objektům vzájemnou komunikaci.
 - Asociované objekty si mohou vzájemně zasílat zprávy (tj. volat svoje metody, přistupovat ke svým atributům atp.)
- Základní charakteristikou všech vztahů je násobnost (kardinalita):
 - **1:1** – každá instance jedné třídy je spojena s nejvýše jednou instancí jiné třídy
 - **1:N (M:1)** – každá instance první třídy spojená s libovolným počtem instancí druhé třídy
 - **M:N** – každá instance jedné i druhé třídy může být spojena s libovolným počtem jiných instancí

Vazba 1:1

- Na žádné straně vazby nebude horní mez násobnosti nikdy větší než 1.
- Diagram tříd popisuje následující situaci:
 - Občan může mít přidělen nejvýše jeden občanský průkaz. Nemusí mít ale žádný
 - Každý občanský průkaz musí být přidělen nějakému občanovi. Nemůže tedy existovat občanský průkaz bez přiděleného občana.



Pozn: Zatímco na jedné straně lze přiřazení protějšku provést kdykoliv v průběhu života objektu (0..1), druhá uvedená násobnost (1) jasně stanovuje podmínku, že protějšek musí být přidělen hned při vzniku objektu.

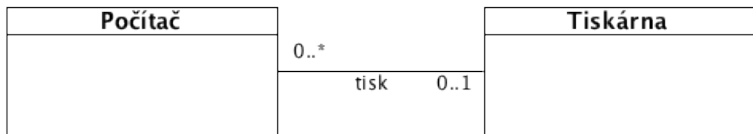
Vazba 1:N/M:1 (1/2)

- První příklad ilustruje situaci, kdy programuji evidenci obyvatel.
- Potřebuji zajistit, aby z každé adresy představující trvalé bydliště mohly vést odkazy na libovolné množství obyvatel.
- Musím tedy evidovat určitý seznam ukazatelů na různé obyvatele.
- Dále jsem namodeloval podmínku, že daný obywatel má pouze jedno trvalé bydliště.



Vazba 1:N/M:1 (2/2)

- Druhý příklad popisuje situaci, kdy v kanceláři je jedna síťová tiskárna a řada počítačů.
 - Počítače využívají tuto sdílenou tiskárnu.
 - Počítač musí mít možnost přistoupit k tiskárně, aby na ní zadal tisk.
 - Tiskárna s počítači naopak komunikovat nemusí (nemůže).



- Vazba bude realizována z počítače směrem k tiskárně.
 - u počítače bude ukazatel na jednu instanci třídy tiskárna (bude nebo nebude inicializovan

Vazba M:N

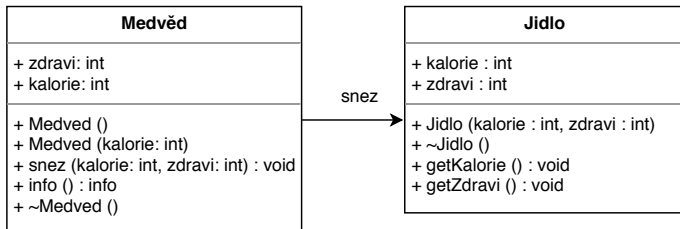
- Princip je zřejmý, nicméně z implementace je ve většině programovacích jazyků nemožná.
- Příklad: Každý vlastník může vlastnit libovolné množství pozemků, každý pozemek (jeho podíl) může být vlastněn libovolným množstvím vlastníků.



- Na první pohled se problém jeví jako snadno řešitelný. V obou třídách budeme mít seznam ukazatelů na protějšky.
- Uvažujme ale situaci, že každý vlastník musel za pozemek zaplatit určitou částku. Každý podíl pozemku stojí jiné množství peněz a každý vlastník za různé pozemky zaplatil různé částky. Cena tedy nemůže být evidována ani u pozemku, ani u vlastníka.

Implementace vazeb – asociace

- Asociace je nejvolnějším typem vazby.
 - objekty existují nezávisle na sobě
 - objekty se propojují jen na krátkou dobu (obvykle po dobu zavolání metody)
- Je využívána zejména pro předání hodnot mezi jednotlivými objekty
- Příklad



Implementace vazeb – agregace

- Agregace (zahrnutí, obsažení) je forma vazby, která popisuje dlouhodobý vztah objektů
- Příkladem agregace může být vztah **Učitel** – **Žák**
 - Každý žák musí mít přiřazeného učitele. Ten bude právě jeden.
 - Učitel může učit jednoho a více studentů.
 - Žák může mít souseda, a to nejvýše jednoho.
 - Žák nemusí mít souseda.
 - Tím sousedem je opět nějaký žák.
- Zřejmě
 1. Učitel i žáci existují nezávisle na sobě. Není zde rozhodně žádná závislost ve smyslu: když vznikne nový žák, vznikne i jeho spolužák, případně učitel. Ani opačně: když zanikne učitel (odejde), nezaniknou žáci, atp. V tomto ohledu není nutné nic řešit.
 2. Vazby jsou delší než zavolání jedné metody. Spolužáci jsou spolužáky do té doby, než je jeden z nich zrušen nebo se přesadí. definované vazby je tedy nutné dlouhodobě ukládat.

Implementace vazeb – kompozice

- Kompozice umožňuje sestavení komplexního objektu z několika dílčích objektů.
- Veškeré objekty jsou spolu pevně svázány a nemohou bez sebe existovat.
- Příkladem agregace může být vztah `Auto` – `Motor`
 - Konstruktor auta musí zajistit vytvoření motoru a destruktore jeho zrušení.
 - Jedině tak bude splněna podmínka, aby motor byl nedílnou součástí auta.