PB161 Programování v jazyce C++
Přednáška 7

Statické položky tříd
Základy OOP

Nikola Beneš

6. listopadu 2018

Klíčové slovo `static`

Znáte z C

Klíčové slovo `static`

Znáte z C

- statické proměnné uvnitř funkcí
- uchovávají si hodnotu mezi voláními funkce
- (schované globální proměnné)

Statické položky (atributy, metody) tříd

- patří třídě jako takové, ne jejím objektům
- je možno k nim přistupovat i bez aktuálního objektu
- mohou k nim přistupovat i objekty
- inicializace statických atributů
  - ne v hlavičkovém souboru (proč?)
  - mimo deklaraci třídy

`lecture07_01.cpp`

Klíčové slovo `static`

Znáte z C

- statické proměnné uvnitř funkcí
- uchovávají si hodnotu mezi voláními funkce
- (schované globální proměnné)

Statické položky (atributy, metody) tříd

- patří třídě jako takové, ne jejím objektům
- je možno k nim přistupovat i bez aktuálního objektu
- mohou k nim přistupovat i objekty
- inicializace statických atributů
  - ne v hlavičkovém souboru (proč?)
  - mimo deklaraci třídy
- od C++17 můžeme psát `inline` i ke statickým atributům, tyto pak můžou být inicializovány přímo uvnitř deklarace třídy
  https://en.cppreference.com/w/cpp/language/static

`lecture07_01.cpp`

Programovací jazyky

Paradigmata (hrubé rozdělení)

- deklarativní
  - logické programování
  - funkcionální programování
  - a jiné (SQL, constraint programming, ...)
- imperativní
  - procedurální (C, Pascal, ...)
  - objektově orientované
    - čisté OOP (posílání zpráv; Objective C, ...)
    - třídy a metody (C++, Java, C#, ...)
    - prototypy (Javascript, ...)
- a mnohá další ...

https://en.wikipedia.org/wiki/Programming_paradigm

Programovací jazyky

Jedno paradigma?

- moderní programovací jazyky jsou typicky multiparadigmatické
- C++: paralelní, funkcionální, generické, metaprogramování, imperativní, objektově orientované (založené na třídách)

https://en.wikipedia.org/wiki/List_of_programming_languages_by_type

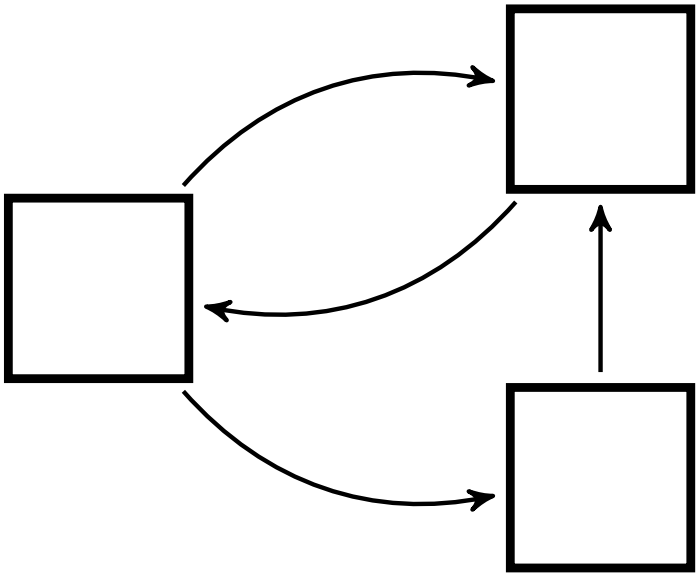
https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages

Nezapomínejte:

- není žádné jedno ideální paradigma
- stejně jako není žádný jeden ideální jazyk

Základní představa OOP

- svět se skládá z objektů
- objekty mají svůj vnitřní stav, který není vidět
- objekty komunikují pomocí zpráv



- objekty se často vytvářejí podle vzoru (třída, prototyp, ...)

Objektově orientované programování

Klasické (čisté) OOP

- Smalltalk, Objective C, (Python)
- všechno je objekt (*včetně tříd*); objekty si posílají zprávy
- můžeme napsat objekt, který dokáže přijmout libovolnou zprávu
- objekty mohou přeposílat zprávy jiným objektům
- objekty mohou měnit své chování (reakci na zprávy) za běhu

OOP ve stylu C++/Java/C#

- třída je datový typ; objekt je instance třídy (hodnota daného typu)
- třídy definují metody; posílání zpráv je volání metod
- rozhraní (seznam metod) třídy je statické (nemění se za běhu)

Principy OOP

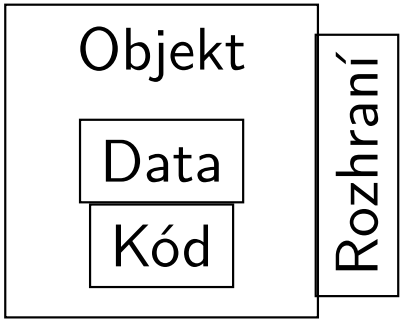
Zapouzdření

Abstrakce

Dědičnost

Polymorfismus

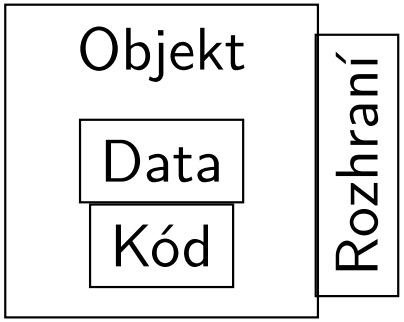
Principy OOP – Zapouzdření (Encapsulation)



- data i kód na jednom místě
- objekt komunikuje jen skrze rozhraní
- umožňuje skrýt vnitřní reprezentaci dat

Výhody:

Principy OOP – Zapouzdření (Encapsulation)



- data i kód na jednom místě
- objekt komunikuje jen skrze rozhraní
- umožňuje skrýt vnitřní reprezentaci dat

Výhody:

- možnost změnit vnitřní implementaci
- ochrana proti chybám
- nutí rozvrhnout program do nezávislých částí
- umožňuje další OOP vlastnosti

Principy OOP – Abstrakce

- souvisí se zapouzdřením
- umožňuje pracovat s ideální představou, nezatěžuje programátora implementačními detaily

Datová abstrakce

- použití dat bez znalosti skutečného umístění/reprezentace
- příklad: soubor na disku, na serveru, ...

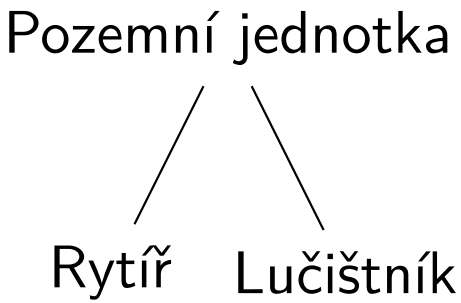
Funkční abstrakce

- uživatel nemusí znát detaily implementace
- příklad (strategická hra): zpráva „zaútoč“ pro různé druhy jednotek

Principy OOP – Dědičnost

- objekt (třída) může dědit od jiného objektu (třídy)
- vztah předek–potomek (podtyp–nadtyp)
- může snižovat opakování kódu; umožňuje polymorfismus

Příklad:



Liskovové princip nahraditelnosti: potomek může zastoupit předka

Vícenásobná dědičnost, kompozice: příště

Principy OOP – Polymorfismus

- souvisí s abstrakcí
- umožňuje psát kód obecně, pouze se znalostí rozhraní
- polymorfismus skrze dědičnost

Příklad: poslat všem pozemním jednotkám ve vybrané oblasti stejný povel

- existují i jiné způsoby realizace polymorfismu

Polymorfismus

Obecný princip polymorfismu

- možnost psát obecný kód, který funguje s různými typy objektů

Různé realizace polymorfismu

Polymorfismus

Obecný princip polymorfismu

- možnost psát obecný kód, který funguje s různými typy objektů

Různé realizace polymorfismu

- přetěžování funkcí
  - *ad-hoc polymorfismus*
- generické programování (C++: šablony, jiné jazyky: generika, ...)
  - *parametrický polymorfismus*
  - ve funkcionálním programování: často jediný druh polymorfismu
- dědičnost
  - *podtypový polymorfismus* (subtype polymorphism)

Realizace OOP v C++

- přístup založený na třídách (`class`, `struct`)
  - členské proměnné tříd – atributy
  - členské funkce – metody
- přístupová práva
  - `public` – vidí všichni
  - `private` – vidí jen objekty dané třídy (+ přátelé)
  - `protected`
    - vidí objekty dané třídy a jejích potomků (+ přátelé)
- implicitní přístupová práva
  - `class` – `private`
  - `struct` – `public`

Realizace OOP v C++

Dědičnost (inheritance)

- třída v C++ může dědit od jiné třídy

```
class Animal { /* ... */ };

class Dog : public Animal { /* ... */ };
```

- co to znamená?

Realizace OOP v C++

Dědičnost (inheritance)

- třída v C++ může dědit od jiné třídy

```
class Animal { /* ... */ };

class Dog : public Animal { /* ... */ };
```

- co to znamená?
  - všechny atributy `Animal` jsou i atributy `Dog` (jaký je k nim přístup?)
  - všechny metody `Animal` jsou i metody `Dog` (ale co když chceme jiné chování?)

Realizace OOP v C++

Dědičnost a přístupová práva

- metody potomka (`Dog`) mají možnost přistupovat pouze k `public` a `protected` položkám předka (`Animal`)
- z vnějšku je možno přistupovat ke všem `public` položkám potomka, ať už jsou definovány v potomkovi nebo předkovi

Jiná než veřejná dědičnost

- pro zajímavost, v tomto předmětu nebudeme používat
- `protected` dědičnost $\rightarrow$ všechny `public` položky předka se změní na `protected`; podobně `private`
- není vztah podtyp – nadtyp (tj. *IS-A*; více příště)
- http://www.gotw.ca/publications/mill06.htm

Realizace OOP v C++

Co bude výstupem?

```
class Animal {
public:
    void speak() const { std::cout << "<generic sound>\n"; }
};

class Dog : public Animal {
public:
    void speak() const { std::cout << "Whoof\n"; }
};

int main() {
    Dog laika;
    Animal& animal = laika;
    laika.speak();
    animal.speak();
}
```

lecture07_02.cpp

Časná a pozdní vazba

Časná vazba (early binding)

- klasické volání funkcí a metod
- překladač ví při překladu, kterou metodu má volat
- v kódu (assembleru) je přímo adresa funkce
- https://godbolt.org/g/nDqasd

Pozdní vazba (late binding)

- která metoda se má volat, se zjistí až za běhu
- v C++ klíčové slovo `virtual`
- každá třída má tabulku virtuálních funkcí
- každý objekt si s sebou nese ukazatel do správné tabulky
- https://godbolt.org/g/Tuf8Cd

Zkuste si zde: https://godbolt.org/g/xBV5b7

Tabulka virtuálních metod

Časná a pozdní vazba

Časná vazba	Pozdní vazba
metoda známa v době překladu	metoda známa až za běhu
v asm kódu přímo adresa funkce	nutno se podívat do `vtable`
rychlejší	pomalejší
v C++ implicitní způsob	v C++ klíčové slovo `virtual`

Různé využití tříd/objektů v C++

Polymorfní třídy	Hodnotové třídy	RAII třídy
virtuální metody	inline metody	inline metody
objekty s identitou	bez identity	objekty spravují zdroje
chování	(interní) stav	(externí) stav
typicky se nekopírují	typicky se kopírují	typicky se nekopírují,
ani nepřesouvají	a přesouvají	ale přesouvat se mohou
často se alokují	typicky se používají	téměř vždy se používají
na haldě	lokálně	lokálně

- OOP se zabývá třídami v levém sloupci
- (toto není nutně vyčerpávající klasifikace všech typů tříd v C++)

pro zajímavost: https://www.youtube.com/watch?v=_cwVvBsZE6M

Kvíz

https://kahoot.it

```
class A {
public:
    void f() { cout << "Af "; }
    virtual void g() { cout << "Ag "; }
};
class B : public A {
public:
    void f() { cout << "Bf "; }
    virtual void g() override { cout << "Bg "; }
};

int main() {

}
```

lecture07_03.cpp

Kvíz

https://kahoot.it

```
class A {
public:
    void f() { cout << "Af "; }
    virtual void g() { cout << "Ag "; }
};
class B : public A {
public:
    void f() { cout << "Bf "; }
    virtual void g() override { cout << "Bg "; }
};

int main() {
    A a; a.f(); a.g(); // 1
}
```

lecture07_03.cpp

Kvíz

https://kahoot.it

```
class A {
public:
    void f() { cout << "Af "; }
    virtual void g() { cout << "Ag "; }
};
class B : public A {
public:
    void f() { cout << "Bf "; }
    virtual void g() override { cout << "Bg "; }
};

int main() {
    B b; b.f(); b.g(); // 2
}
```

lecture07_03.cpp

Kvíz

https://kahoot.it

```
class A {
public:
    void f() { cout << "Af "; }
    virtual void g() { cout << "Ag "; }
};
class B : public A {
public:
    void f() { cout << "Bf "; }
    virtual void g() override { cout << "Bg "; }
};

int main() {
    A a; A& refa = a; refa.f(); refa.g(); // 3
}
```

lecture07_03.cpp

Kvíz

https://kahoot.it

```
class A {
public:
    void f() { cout << "Af "; }
    virtual void g() { cout << "Ag "; }
};
class B : public A {
public:
    void f() { cout << "Bf "; }
    virtual void g() override { cout << "Bg "; }
};

int main() {
    B b; A& refb = b; refb.f(); refb.g(); // 4
}
```

lecture07_03.cpp