C++; delete Java;

Část 6: smart pointery

Kennny

srpen 2017

Motivace

- eliminovat memory leaky
- pracovat s pamětí bezpečně
- využít moderní principy OOP
- vyhnout se garbage collectoru

„If Java had true garbage collection, most programs would delete themselves upon execution.“

Řešení

- smart pointery
- v zásadě existují tři (šablonové) typy:
  - `unique_ptr`
  - `shared_ptr`
  - `weak_ptr`
- dříve ještě `auto_ptr`, nyní deprecated

Smart pointery

- reference counting
- RAII struktury
  - konstruktor zvyšuje čítač
  - destruktor snižuje
- podporují vlastnosti OOP (dědičnost, polymorfismus, ..)

std::unique_ptr

- unikátní ukazatel - ve smyslu vlastníka
- takový objekt má pouze jednoho vlastníka
- nedá se kopírovat, pouze přesunout (zneplatní původní)
- přetížen operátor `->` pro přístup k paměti, aby se choval jako každý jiný pointer
- lze vyzvednout surový pointer metodou `get()`
- vytváření pomocí `std::make_unique` s parametry konstruktoru

```
std::unique_ptr<Coords> a = std::make_unique<Coords>(5.5, 2.5);
// analogie k puvodnimu
Coords* b = new Coords(5.5 2.5);
```

Příklad

- Prostor pro příklad 06_a_unique_ptr

std::shared_ptr

- "silný" ukazatel
- reference counting v plné podobě
- také přetížen operátor `->`
- lze vyzvednout surový pointer metodou `get()`
- vytváření pomocí `std::make_shared` s parametry konstruktoru

```
std::shared_ptr<Coords> a = std::make_shared<Coords>(5.5, 2.5);
// analogie k puvodnimu
Coords* b = new Coords(5.5 2.5);
```

std::shared_ptr

- již lze kopírovat

```
std::shared_ptr<Coords> a = std::make_shared<Coords>(5.5, 2.5);
std::shared_ptr<Coords> b = a;
std::shared_ptr<Coords> c = b;
```

- všechny ukazují na stejné místo, reference counter je roven 3
- jakmile se zavolají destruktory, sníží se counter na 0 a poslední destruktor ho dealokuje
- asi nejběžnější varianta smart pointeru

Příklad

- Prostor pro příklad 06_a_shared_ptr

std::weak_ptr

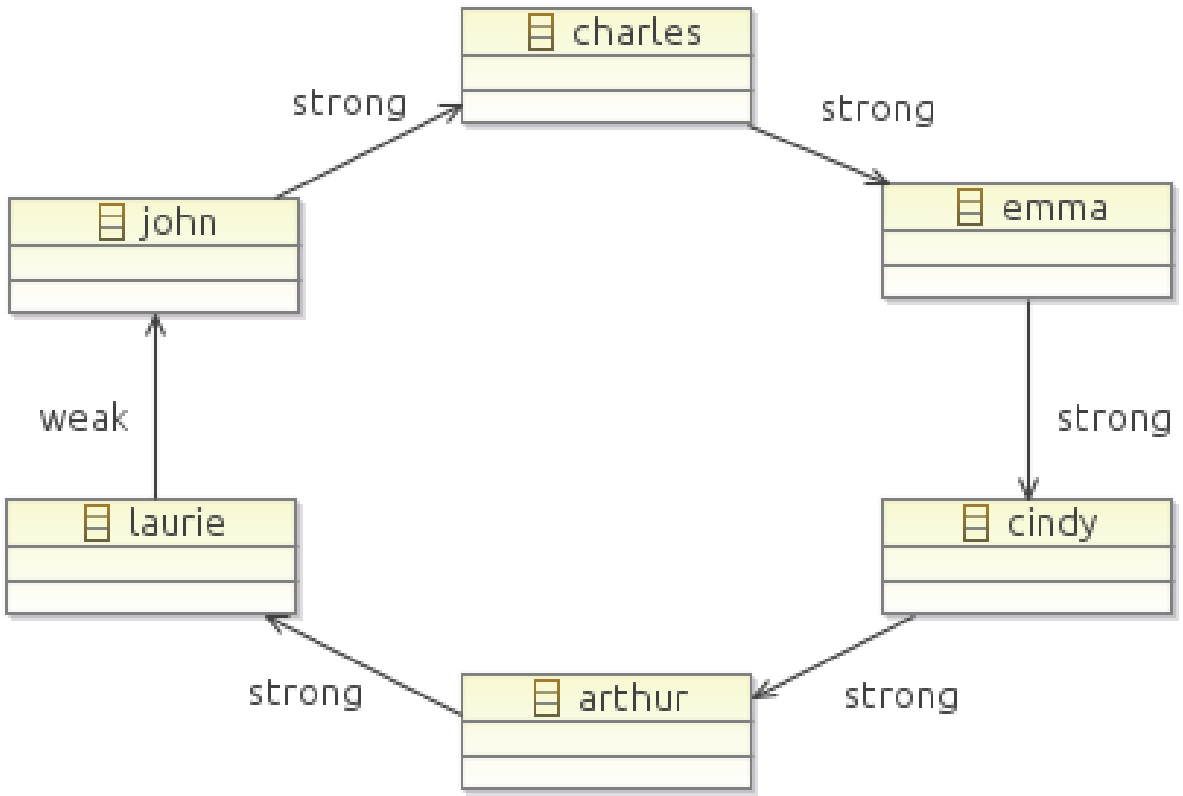
- "slabý" ukazatel, který se nepodílí na reference countingu
- sdružený se shared pointerem
- nemá přetížen operátor `->`
- inicializuje se přiřazením shared pointeru
- před použitím nutno konvertovat na shared pointer

```
std::weak_ptr<Coords> a = sharedPtrCoords;

if (std::shared_ptr<Coords> sptr = a.lock())
    sptr->DoSomething();
```

- pokud již byla paměť smazána, `lock()` vrátí `nullptr`
- hodí se pro implementaci kruhových závislostí

std::weak_ptr



Obrázek: Kruhová závislost, ukradeno z `https://visualstudiomagazine.com/articles/2012/10/19/circular-references.aspx`

Příklad

- Prostor pro příklad 06_a_weak_ptr

Konec 6. části

```
exit(0);
```