C++; delete Java;

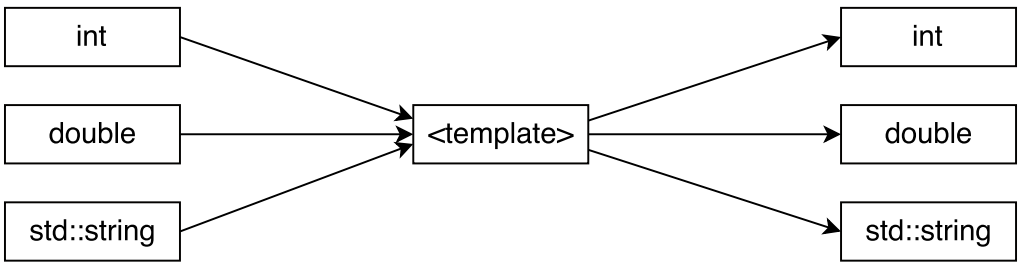
Část 9: šablony

Kennny

srpen 2017

Šablony

- princip, který dovoluje vytvářet více variant implementace z jedné předlohy
- implementační "mezikrok" generické implementace
- nemělo by nahrazovat dědičnost, polymorfismus a další principy OOP



Šablony

- šablonová může být funkce (metoda) a třída
- klíčové slovo `template` a `typename`
- přibyde symbol, kterým nahrazujeme typ

```
template<typename T>
void DoSomething(T& input)
{
    // ...
}
```

- při volání buď kompilátor rozhodne, co za typ dosadí, nebo můžeme explicitně uvést

```
DoSomething(5); // pravdepodobne T = int
DoSomething<double>(5); // T = double
```

Šablony

- při překladu se šablona specializuje tak, aby vyhověla všem voláním

```
void DoSomething(int& input)
{
    // ...
}

void DoSomething(double& input)
{
    // ...
}
```

Explicitní specializace

- pokud nechceme specializaci nechat na kompilátoru, můžeme specializovat explicitně

```
template<>
void DoSomething<double>(double& input)
{
    // ...
}
```

- kompilátor pak nebude vytvářet novou specializaci, ale použije naši

Jednoduchý příklad

- příkladem použití může být například potřeba vytisknout něco v řádce - víme že budeme pracovat se spoustou typů, nemůžeme tedy implementovat X různých variant
- zároveň ale víme, že std::cout se už umí chovat podle vstupního typu

```
template<typename T>
void printLine(T& input)
{
    std::cout << input << std::endl;
}
```

Příklad

- Prostor pro příklad 09_a_templates

Konec 9. části

```
exit(0);
```