Programowanie w C++

Kurs (średnio)zaawansowany

Zajęcia numer 19

Rafał Berdyga
rberdyga@gmail.com

Plan na dzisiaj

- Wskaźnik *this*
- Lista inicjalizacyjna
- Relacje między klasami - relacja agregacji i kompozycji
- Diagramy klas – wprowadzenie
- Podstawy obsługi rozproszonego systemu kontroli wersji: Git

Wskaźnik *this*

W zasięgu lokalnym definicji każdej metody klasy dostępny jest wskaźnik *this*.

Jest to wskaźnik do obiektu tej klasy, której składową jest ta metoda.

Gdy mamy np. zdefiniowaną klasę **`Student`** to w treści jej metod wskaźnik this jest typu **`Student*`**

Z kolei gdy używamy klasy **`Teacher`** to tu wskaźnik jest typu **`Teacher*`**

Wskaźnik *this* – inicjalizacja atrybutów

```
#include <iostream>
using namespace std;

class Osoba
{
    int wiek_;
    int wzrost_;
    string  imie_;

public:
    Osoba (int wiek, int wzrost, string imie)
    {
        this -> wiek_ = wiek;
        this -> wzrost_ = wzrost;
        this -> imie_ = imie;
    }
};
```

- Wskaźnik *this* jest automatycznym wskaźnikiem na konkretny obiekt danej klasy (TEN OBIEKT),
- Każdy tworzony obiekt ma swój własny wskaźnik *this*.

Wskaźnik *this* ma szerokie zastosowanie, dlatego warto się z nim oswoić. Ponadto występuje on także w innych językach programowania jak np. Java, C#.

Lista inicjalizacyjna

Lista inicjalizacyjna jest rozszerzeniem możliwości zwykłego konstruktora. Jej zadaniem jest **inicjalizacja** składowych nowego obiektu. Ważnym jest fakt, że wykonuje się ona jeszcze **zanim obiekt zacznie istnieć.**

```
class Osoba
{
    int wiek_;
public:
    Osoba (int wiek)
    {
        wiek_ = wiek;
    }
};
```

Konstruktor parametryczny

```
class Osoba
{
    int wiek_;
public:
    Osoba (int wiek) : wiek_(wiek)
    {}
};
```

Konstruktor parametryczny

Lista inicjalizacyjna

Lista inicjalizacyjna cd.

Cechą konstruktorów jest to, że wykonują się one w momencie kiedy obiekt klasy już istnieje. Co za tym idzie, **konstruktory** mogą modyfikować wartości składowych klas jednak w niektórych przypadkach staje się to niemożliwe.

Jeżeli składową klasy jest zmienna **const** wtedy nie będziemy mieli możliwości nadania jej wartości poprzez konstruktor.

Lista inicjalizacyjna znajduje się w definicji konstruktora i poprzedzona jest dwukropkiem. Argumenty w **liście inicjalizacyjnej** przypisywane są składowym klasy w następujący sposób:

```
atrybut1(argument1), atrybut2(argument2)
```

Lista inicjalizacyjna – składowe const

```
#include <iostream>
using namespace std;

class Osoba
{
    const int wiek_;
    const int wzrost_;
    string  imie_;

public:
    Osoba (int wiek, int wzrost, string imie)
    {
        // blad kompilacji !!!
        this -> wiek_ = wiek;
        this -> wzrost_ = wzrost;
        this -> imie_ = imie;
    }
};
```

Tak jak było napisane na poprzednim slajdzie, w konstruktorze nie można przypisać zmiennej *const* żadnej wartości, ponieważ podczas wywołania konstruktora **obiekt już istnieje**.

Lista inicjalizacyjna – składowe const cd.

```
#include <iostream>
using namespace std;

class Osoba
{
    const int wiek_;
    const int wzrost_;
    string  imie_;

public:
    Osoba (int wiek, int wzrost, string imie) : wiek_(wiek), wzrost_(wzrost)
    {
        this -> imie_ = imie;
    }
};
```

W przypadku użycia **listy inicjalizacyjnej** ciało konstruktora może być puste (ale można oczywiście umieścić w nim jakieś instrukcje). Niektóre składowe mogą być **inicjalizowane** poprzez konstruktor a inne poprzez **listę inicjalizacyjną**.

Składowe `wiek` oraz `wzrost` musimy zainicjalizować na liście (są one opatrzone specyfikatorem **const**), jednak wartość składowej `imie` można przypisać w konstruktorze.

OBOWIĄZKOWA praca domowa

Napisz dwie dowolne klasy, posiadające minimum 3 atrybuty każda. Zdefiniuj w każdej z nich trzy konstruktory: bezparametrowy, parametryczny i kopiujący.

- W pierwszej własnej klasie proszę zainicjować atrybuty przy użyciu wskaźnika *this* w każdym z trzech konstruktorów.

- W drugiej własnej klasie proszę zainicjować atrybuty przy użyciu list inicjalizacyjnych przy każdym z trzech konstruktorów.

Wszystkie informacje na temat użycia tych technik (wraz z przykładami) są dostępne na wcześniejszych slajdach.

Kod oddajemy poprzez utworzenie repozytorium na stronie GitHub oraz **zaproszenie do wglądu w repozytorium prowadzącego.**

Proszę przed oddaniem doprowadzić kod do stanu, w którym się on kompiluje!

Kompozycja obiektów (OOP)

W programowaniu obiektowym istnieje potrzeba stworzenia **kompozycji obiektów** przed rozpoczęciem pisanie kodu.

Na przykład aby wymodelować świat gry komputerowej musimy umieścić w nim wiele postaci i przedmiotów (obiektów). Musimy zastanowić się jakich klas obiektów będziemy potrzebować i jak będą one ze sobą współdziałać.

Kompozycja obiektów to logiczna albo koncepcyjna struktura przechowywania informacji. Nie należy mylić jej z implementacją albo fizyczną strukturą danych.

Projektowanie na **poziomie konceptualnym** jest pierwszym etapem przy tworzeniu większych systemów.

Dopiero następnym etapem jest projektowanie logiczne (struktury danych), później implementacja (pisanie kodu), testowanie itd…

Kompozycja i agregacja (relacje)

Piszemy program, w którym musimy określić **relację** pomiędzy Uniwersytetem, Wydziałem a Profesorami.

Uniwersytet składa się z **Wydziałów** (np. wydział Informatyki). Każdy wydział zawiera w sobie kilku lub kilkunastu **Profesorów**.

Jeśli uniwersytet zostanie zamknięty, to wydział już nie może istnieć. Profesorowie tych konkretnych wydziałów jednak wciąż mogą istnieć w naszym wirtualnym świecie.

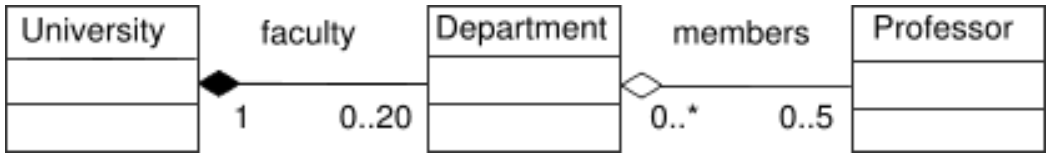
Z punktu widzenia projektowania **Uniwersytet jest kompozycją wydziałów** – wydział nie istnieje bez Uniwersytetu.

Wydział zaś **agreguje** **Profesorów** – są częścią wydziału, ale mogą istnieć bez niego.

Kompozycja i agregacja cd.

```
class Professor;  // zdefiniowana w innym pliku

class Department {
 public:
 Department(const std::string& name): name_(name) {}

 private:
  // Agregacja: Profesorowie mogą istnieć bez wydzialu
  std::vector<Professor> members_;
  const std::string name_;
};

class University{
 public:
University() = default;

 private:
  // Kompozycja: Wydziały istnieją tak długo, jak istnieje uniwersytet
  std::vector<Department> faculty_ = {
      Department("chemistry"),
      Department("physics"),
      Department("arts"),
  };
};
```

O wektorach za tydzień!



Notacja UML

Język do modelowania, wizualizacji i dokumentowania projektów związanych z wytwarzaniem oprogramowania, jak również wielu innych złożonych systemów.