Programowanie obiektowe

Laboratorium 5 i 6 - przypomnienie wiadomości o OOP na przykładzie C++

mgr inż. Krzysztof Szwarc

krzysztof@szwarc.net.pl

Sosnowiec, 29 marca 2017

Struktura klasy

```
class NazwaKlasy
{
    //ciało klasy
};
```

Klasy w $C++$

W $C++$ nie da się bezpośrednio określić modyfikatora dostępu klasy (operacja może zostać wykonana wyłącznie dla klas zagnieżdżonych).

Zalecane podejście do organizacji kodu

Rozdzielenie definicji od implementacji

Zaleca się, aby klasy w $C++$ były rozdzielane na dwa pliki - jeden zawierający definicję oraz drugi jej implementację. Pierwszy z nich powinien zostać zapisany w pliku nagłówkowym z rozszerzeniem *h* (określone w nim zostają pola oraz prototypy metod), a drugi w module programu o rozszerzeniu *cpp* (zawierającym ciało metod).

Struktura pliku nagłówkowego

```
class NazwaKlasy
{
    modyfikatorDostepu:
        typPola nazwaPola;
        zwracanyTyp nazwaMetody(parametry);
};
```

Struktura pliku z implementacją

```
#include "NazwaKlasy.h"

typ NazwaKlasy::nazwaMetody(parametry){
    return typ;
}
```

Przykładowa zawartość pliku nagłówkowego

```
class Czlowiek
{
    public:
        Czlowiek();
        virtual ~Czlowiek();
        void przedstawSie();

    private:
        int wiek;
};
```

Przykładowa zawartość pliku z implementacją

```
#include <iostream>
#include "Czlowiek.h"

Czlowiek::Czlowiek() {}

Czlowiek::~Czlowiek() {}

void Czlowiek::przedstawSie()
{
    std::cout<<"Człowiek";
}
```

Klasy

Modyfikator dostępu

W $C++$ wyróżniamy trzy modyfikatory dostępu:

1. public - dostęp do niego ma każdy obiekt.
2. private - dostęp do niego ma wyłącznie właściciel (domyślny modyfikator dostępu).
3. protected - dostęp do niego ma właściciel oraz klasy dziedziczące po nim (klasy pochodne).

Struktura pliku nagłówkowego - konstruktory

```
class NazwaKlasy
{
    public:
        NazwaKlasy();
        NazwaKlasy(int par);
        NazwaKlasy(NazwaKlasy& kopia);
        NazwaKlasy(InnaKlasa& obiekt);
        virtual ~NazwaKlasy();
        /* virtual zapewnia wywołanie
           wszystkich destruktorów klas
           pochodnych */
};
```

Struktura pliku z implementacją - konstruktory

```
#include "NazwaKlasy.h"

NazwaKlasy::NazwaKlasy(){}

NazwaKlasy::~NazwaKlasy(){}

NazwaKlasy::NazwaKlasy(int par){}

NazwaKlasy::NazwaKlasy(NazwaKlasy& kopia){}

NazwaKlasy::NazwaKlasy(InnaKlasa& obiekt){}
```

Lista inicjalizacyjna

Definicja

Lista inicjalizacyjna stanowi rozszerzenie możliwości konstruktora i umożliwia inicjalizację składowych nowego obiektu. Przykład dla pliku zawierającego implementację klasy.

```
#include "NazwaKlasy.h"

NazwaKlasy::NazwaKlasy() : pole(5), pole2(5)
{
    pole = 6;
    /* wystarczy wyłącznie jeden z zapisów - po
       utworzeniu obiektu wartość pola będzie
       wynosić 6 (najpierw zostanie ustawiona
       wartość 5, a następnie nadpisana szóstką) */
}
```

Tworzenie obiektów - stos, a sterta

W $C++$ możemy tworzyć obiekty zarówno na stosie (1), jak i na stercie (2). W pierwszym przypadku odwołujemy się do pól oraz metod za pośrednictwem kropki, a w drugim korzystamy z konstrukcji $\rightarrow$.

```
(1)
Klasa obiekt;
std::cout<<obiekt.pole;
(2)
Klasa* obiekt = new Klasa();
std::cout<<obiekt->pole;
delete obiekt;
```

Dziedziczenie - plik nagłówkowy

```
class NazwaKlasyPochodnej :
    modyfikatorDostepu NazwaKlasyBazowej,
    modyfikatorDostepu NazwaKolejnejKlasy
{
    // ciało klasy
};
```

Klasa pochodna dziedziczy prawie wszystkie składowe klasy bazowej (za wyjątkiem konstruktorów, destruktorów i przeciążonych operatorów przypisania). Przy zastosowaniu modyfikatora dostępu public, w klasie pochodnej zostanie zachowana taka sama widoczność składowych klasy, jaka obowiązywała w klasie bazowej.

Zadania

1. Napisz klasę Czlowiek zawierającą publiczną metodę przedstawSie, która wypisze tekst „Człowiek”.
2. Napisz klasę Pracownik zawierającą publiczną metodę przedstawSie, która wypisze tekst „Pracownik”. Niech klasa dziedziczy po klasie Czlowiek.

Wczesne wiązanie

```
Pracownik* pracownik = new Pracownik();
Czlowiek* czlowiek = new Czlowiek();
czlowiek->przedstawSie(); // Człowiek
pracownik->przedstawSie(); // Pracownik
czlowiek = pracownik;
czlowiek->przedstawSie(); // Człowiekek
```

Polimorfizm nie zadziałał tak jak chcieliśmy - $C++$ skorzystał z wczesnego wiązania i na etapie kompilacji została wybrana metoda należąca do klasy, której odpowiadał typ wskaźnika czlowiek.

Wiązanie dynamiczne

Aby kompilator wywołał metodę klasy, na którą wskazuje wskaźnik należy użyć słowa kluczowego virtual przed deklaracją metody (wystarczy dopisać je w klasie bazowej - we wszystkich klasach pochodnych metoda będzie wiązana dynamicznie).

```
class Czlowiek
{
    public:
        Czlowiek();
        virtual ~Czlowiek();
        virtual void przedstawSie();

    private:
        int wiek;
};
```

Wiązanie dynamiczne - przykład

```
Pracownik* pracownik = new Pracownik();
Czlowiek* czlowiek = new Czlowiek();
czlowiek->przedstawSie(); // Człowiek
pracownik->przedstawSie(); // Pracownik
czlowiek = pracownik;
czlowiek->przedstawSie(); // Pracownik
```

Część praktyczna

1. Wykonaj zadania ze stron 70-73 ze skryptu: `http://w.s.w.w.interia.pl/skrypt.pdf`

Dziękuję za uwagę