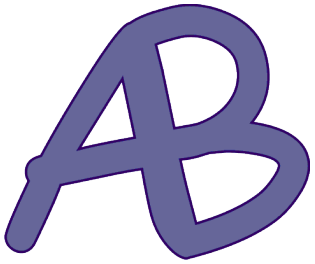
Programowanie obiektowe

dr Artur Bartoszewski
Katedra Informatyki
UTH Radom



Wykład 5: Klasy cz. 3

Przeciążanie metod w klasach

Przeciążanie metod

Przeciążanie metod (ang. Overloading) - pozwala na tworzenie metod o takich samych nazwach, ale różniących się listą parametrów, tzn. liczbą lub typem parametrów z którymi metoda jest wywoływana.

```
1  class Klasa
2  {
3    public:
4      void funkcja();
5      void funkcja(int a);
6      void funkcja(float a);
7      void funkcja(int a, int b);
8  };
```

Nieprawidłowe jest:

- utworzenie w jednej klasie dwóch metod o identycznej nazwie i przyjmującej takie same parametry,
- metod o takiej samej nazwie i parametrach, ale różniące się **tylko** zwracanym typem.

Przeciążanie metod

```
class dodawanie{
   int dodaj(int a, int b){
      return a+b;
   }

   double dodaj(double a, double b){
      return a+b;
   }
}
```

```
class dodawanie{
   int dodaj(int a, int b){
      return a+b;
   }

   double dodaj(int a, int b){
      return a+b;
   }
}
```

Konstruktor i destruktory
(część II)

Konstruktor i destruktor

Jawne wywołanie konstruktora

Obiekt może być też stworzony przez jawne wywołanie konstruktora. W efekcie otrzymujemy obiekt, który nie ma nazwy, a czas jego życia ogranicza się do wyrażenia, w którym go użyto.

```
nazwa_klasy(argumenty)
```

Uwaga: nie stosujemy zapisu:

```
~~obiekt.funkcja_składowa(argumenty)~~
```

Konstruktor nie jest wywoływany na rzecz jakiegoś obiektu, bo ten obiekt jeszcze nie istnieje. Zadaniem konstruktora jest go utworzyć.

Konstruktor i destruktor

Jawne wywołanie destruktora

Należy podać całą jego nazwę. Jawne wywołanie destruktora nie może się zacząć od ~(wężyka) i wcześniej musi być albo obiekt, na rzecz którego jest wywoływany i kropka lub wskaźnik do obiektu „->”

```
obiekt.~klasa();
```

```
wskaznik->~klasa();
```

Konstruktor i destruktor

Konstruktor domyślny

Konstruktor domyślny to konstruktor, który można wywołać bez żadnego argumentu.

```
class klasa {
    public :
        klasa();  //konst.  domyślny
    };
```

Konstruktor i destruktor

Konstruktor domyślny

Konstruktor domyślny może posiadać dowolną ilość parametrów, jednak wszystkie muszą mieć zdefiniowaną **wartość domyślną** (aby nie trzeba było podawać ich wartości podczas inicjacji klasy).

```
class klasa {
    public :
     klasa (int a=5, char *s = NULL);
     // to też może być konst. domniemany
    };
```

Konstruktor i destruktor

Konstruktor domyślny z parametrami domyślnymi

```
5   class Osoba {
6   public:
7       string imie;
8       int wiek;
9
10      Osoba(string imie = "NN", int wiek = 0) {
11          this->imie = imie;
12          this->wiek = wiek;
13      }
14  };
15
16  int main()
17  {
18      Osoba *ktos = new Osoba();
19      Osoba ktos2("Karol", 22);
20      cout << ktos->imie << endl; //wyswietli NN
21      cout << ktos2.imie << endl;   //wyswietli Kar
22      return 0;
23  }
```

Konstruktor i destruktor

Przeciążanie konstruktorów

Jedna klasa może posiadać kilka konstruktorów różniących się lista parametrów (ich liczbą i/lub typem).

```
class klasa {
   public :
    klasa(void) ; //konst.  domniemany
    klasa(int);
    klasa(float)
    klasa(int, int);
   };
```

Konstruktor i destruktor

Przeciążanie konstruktorów

```
class klasa {
   public :
    // klasa(void) ; //konst.  domniemany
    klasa(int);
    klasa(float)
    klasa(int, int);
    klasa (int a=5, char *s = NULL);
    // to też może być konst. domniemany ale tylko jeden w klasie
   };
```

Konstruktor i destruktor

Lista inicjalizacyjna konstruktora

Czasami zachodzi potrzeba zainicjowania zmiennej w trakcie tworzenia klasy, a nie po jej utworzeniu. Korzystamy wtedy z tak zwanej listy inicjalizacyjnej konstruktora.

Lista inicjalizacyjna to lista oddzielonych przecinkami identyfikatorów pól (składowych) z podanymi w nawiasach okrągłych argumentami dla konstruktorów obiektów będących składowymi tworzonego obiektu. Zwykle są to jednocześnie argumenty formalne definiowanego konstruktora, choć nie musi tak być.

Konstruktor i destruktor

Lista inicjalizacyjna konstruktora

```
class Osoba {
public:
    int wiek;

    Osoba(int WIEK) {
        wiek = WIEK;
    }
};
```

konstruktor wieloargumentowy

```
class Osoba {
public:
    int wiek;

    Osoba(int WIEK) : wiek(WIEK) {

    }
};
```

konstruktor wieloargumentowy

lista inicjalizacyjna

p-programowanie.pl

Wartość zmiennej WIEK, która jest **argumentem konstruktora** przypisywana jest zmiennej wiek, która jest składnikiem klasy.
Po przecinku można wypisać kolejne **inicjalizowane** składowe.

Konstruktor i destruktor

Taki zapis ma kilka bardzo istotnych zalet.

1. **Jest szybszy** - różnice są znaczne gdy przyjdzie do wykonywania pomiarów czasowych.
2. **Jest czytelniejszy** - programista nie musi analizować zawartości konstruktora, by wiedzieć jaką domyślną wartością zostanie zainicjowana klasa.
3. Umożliwia inicjowanie zmiennych zdefiniowanych jako **stałe** (const).
4. Umożliwia inicjowanie zmiennych zdefiniowanych jako **referencje** (tylko tą metodą można zainicjować zmienną zadeklarowaną np. tak: int & zmienna;).
5. **Jest metodą stosowaną przy dziedziczeniu klas**.

Konstruktor i destruktor

Lista inicjalizacyjna konstruktora

- ✓ Zwróćmy uwagę, że w przypadku użycia listy inicjalizacyjnej ciało konstruktora jest puste. Można oczywiście umieścić w nim jakiejś instrukcje.
- ✓ Niektóre składowe mogą być inicjalizowane poprzez konstruktor a inne poprzez listę inicjalizacyjną.
- ✓ Jeśli w klasie tylko deklarujemy konstruktor, a jego definicję podajemy poza klasą, to w deklaracji listy inicjalizacyjnej nie umieszczamy.

```
1   #include <iostream>
2   using namespace std;
3   class klasa
4   {
5       int a;
6       char b;
7       std::string c;
8   public:
9       klasa();
10  };
11
12  klasa::klasa(): a(100), b('x'), c("napis")
13  {
14      cout << "a = " << a << endl;
15      cout << "b = '" << b << "'" << endl;
16      cout << "c = \"" << c << "\"" << endl;
17  }
18
19  int main()
20  {
21      klasa obiektKlasy;
22      return( 0 );
23  }
24
```

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Gręboszj J. : ***Symfonia C++, Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zajrzeć także do:

- Sokół R. : ***Microsoft Visual Studio 2012 Programowanie w Ci C++,*** Helion.
- Kerninghan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Gręboszj J. : ***Pasja C++,*** Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne