Politechnika Wrocławska
Wydział Mechaniczny
Katedra Konstrukcji i Badań Maszyn i Pojazdów

Programowanie obiektowe Lab 8

Dr inż. Paweł Maślak

pawel.maslak@pwr.edu.pl

Laboratorium 8: Konstruktory Destruktory Tworzenie obiektów

Definicja klasy

Definicja klasy zaczyna się od słowa kluczowego **class**, po którym następuje nazwa klasy. Wnętrze klasy jest otoczone parą nawiasów klamrowych.Poniżej powszechna definicja klasy:

```
modyfikator_dostepu class nazwa_klasy
{
    // pola klasy
    modyfikator_dostepu typ_danych zmienna1;
    modyfikator_dostepu typ_danych zmienna2;
    ...
    modyfikator_dostepu typ_danych zmiennaN;
    // metody klasy
    modyfikator_dostepu zwracany_typ metoda1(lista_parametrow)
    {
        wnetrze_metody
    }
    modyfikator_dostepu zwracany_typ metoda2(lista_parametrow)
    {
        wnetrze_metody
    }
    ...
    modyfikator_dostepu zwracany_typ metodaN(lista_parametrow)
    {
        wnetrze_metody
    }
}
```

gdzie,

- **modyfikator_dostepu** określa dostępność samej klasy oraz wszystkich jej składowych. Jeżeli modyfikator dostępu nie będzie podany dla klasy to domyślnie będzie to **internal**. Domyślny modifykator dostępu dla składowych klasy to **private**;
- **zwracany_typ** określa rodzaj zmiennej, która zostanie zwrócona z metody o ile typ danych będzie zwracany. Jeżeli nie to zwracany typ takiej metody to **void**;
- aby uzyskać dostęp do składowej klasy należy posłużyć się operatorem kropki (**.**);
- operator kropki (**.**) łączy nazwę obiektu z nazwą składowej.

Definicja klasy

```
using System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            Pudelko pudelko1 = new Pudelko(); // definicja obiektu1 typu Pudelko
            Pudelko pudelko2 = new Pudelko(); // definicja obiektu2 typu Pudelko
            // spefyfikacja 1
            pudelko1.dlugosc = 5.5;
            pudelko1.wysokosc = 6.0;
            pudelko1.szerokosc = 3.0;
            // spefyfikaja 2
            pudelko2.dlugosc = 3.5;
            pudelko2.wysokosc = 2.0;
            pudelko2.szerokosc = 1.0;
            // Objetosc 1
            double obj1 = pudelko1.ObliczObjetosc(pudelko1.dlugosc, pudelko1.szerokosc, pud
elko1.wysokosc);
            Console.WriteLine("Objetość pudełka nr 1: {0}", obj1);
            // Objetosc 2
            // Obliczenie bez użycia powyższej metody
            double obj2 = pudelko2.dlugosc * pudelko2.szerokosc * pudelko2.wysokosc;
            Console.WriteLine("Objetość pudełka nr 2: {0}", obj2);
            Console.ReadKey();
            // Wynik działania programu
            //Objetosc pudelka nr 1: 99
            //Objetosc pudelka nr 2: 7
        }
    }
    class Pudelko
    {
        public double dlugosc;
        public double szerokosc;
        public double wysokosc;
        // Metoda do obliczania objetosci zdefinowa jaka skladowa klasy
        public double ObliczObjetosc(double dl, double szer, double wys)
        {
            return dl * szer * wys;
        }
    }
}
```

Metody klasy oraz hermetyzacja

Metoda klasy, podobnie jak pole klasy, ma swoją definicję. Działa na każdym obiekcie klasy, której jest składową i ma dostęp do wszystkich składowych tej klasy.

Zmienne składowe są atrybutami obiektu (z punkcji widzenia projektowania) i przechowywane są, jako prywatne z punktu widzenia hermetyzacji. Składowe te mogą być dostępne jedynie, jeżeli zostaną opatrzone modyfikatorem dostępu **public**.

Wszystkie powyższe punkty zdecydowanie łatwiej będzie dostrzeć na przykładzie. Spróbujmy zatem ustawić oraz pobrać wartości od różnych składowych klasy:

```
using System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            Pudelko pudelko1 = new Pudelko(); // definicja obiektu1 typu Pudelko
            Pudelko pudelko2 = new Pudelko(); // definicja obiektu2 typu Pudelko
            // spefyfikacja 1
            pudelko1.UstawDlugosc(5.5);
            pudelko1.UstawWysokosc(6.0);
            pudelko1.UstawSzerokosc(3.0);
            // spefyfikaja 2
            pudelko2.UstawDlugosc(3.5);
            pudelko2.UstawWysokosc(2.0);
            pudelko2.UstawSzerokosc(1.0);
            // Objetosc 1
            double obj1 = pudelko1.ObliczObjetosc();
            Console.WriteLine("Objetość pudełka nr 1: {0}", obj1);
            // Objetosc 2
            // Obliczenie bez użycia powyższej metody
            double obj2 = pudelko2.ObliczObjetosc();
            Console.WriteLine("Objetość pudełka nr 2: {0}", obj2);
            Console.ReadKey();
            // Wynik działania programu
            //Objetosc pudelka nr 1: 99
            //Objetosc pudelka nr 2: 7
        }
    }
    class Pudelko
    {
        private double dlugosc; // pole dostepne tylko z wnetrza klasy
        private double szerokosc; // pole dostepne tylko z wnetrza klasy
        private double wysokosc; // pole dostepne tylko z wnetrza klasy
        // definiujemy metody, które mają dostęp do tych pól oraz są publiczne
        public void UstawDlugosc(double dl)
        {
            dlugosc = dl;
        }
        public void UstawSzerokosc(double szer)
        {
            szerokosc = szer;
        }
        public void UstawWysokosc(double wys)
        {
            wysokosc = wys;
        }
        // Metoda do obliczania objetosci w innej postaci niż w pierwszym przykladzie
        // Składowe klasy maja już swoje wartości, mozemy obliczyć objętosc
        public double ObliczObjetosc()
```

```
        {
            return dlugosc * szerokosc * wysokosc;
        }
    }
}
```

Konstruktor klasy

Konstruktor klasy jest specjalną metodą, która jest wykonywana zawsze, kiedy tworzony jest obiekt klasy.

Konstruktor ma dokładnie taką samą nazwę jak nazwa klasy oraz nie zwraca żadnego typu. Poniższa konstrukacja pokazuje przykład użycia konstruktora:

```
using System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            // W momencie tworzenia nowego obiektu dojdzie do wywołania konstruktora
            KlasaZKonstruktorem kzk = new KlasaZKonstruktorem();

            Console.ReadKey();
        }
    }
    class KlasaZKonstruktorem
    {
        private string tekst;
        public KlasaZKonstruktorem()
        {
            tekst = "Wywołanie konstruktora klasy";
            Console.WriteLine(tekst);
        }
    }
}
```

Domyślny konstruktor nie ma żadnego parametru, jeżeli jednak zajdzie taka potrzeba przygotowany przez nas konstruktor te parametry może posidać. Konstruktor taki nosi nazwę konstruktora parametryzowanego. Technika taka pozwala na przypisanie wartości początkowej do obiektu w trakcie jego tworzenia co pokazano na poniższym przykładzie:

```
sing System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            string marka = "";
            Console.Write("Podaj markę samochodu: ");
            marka = Console.ReadLine();
            KonstruktorParametryzowany kp = new KonstruktorParametryzowany(marka);
            Console.ReadKey();
        }
    }
    class KonstruktorParametryzowany
    {
        private string marka;
        public KonstruktorParametryzowany(string parametr)
        {
            marka = parametr;
            Console.WriteLine("Marka podana przez użytkownika to: {0}", marka);
        }
    }
}
```

Destruktor klasy

Destruktor jest specjalną metodą klasy, która jest wykonywana przez Garbage Collector. Destruktor ma dokładnie taką samą nazwę jak nazwa klasy oraz przedrostek (~). Destruktor nie może zwracać oraz przyjmować żadnych parametrów.

Destruktor może być bardzo przydatny, gdy chcemy zwolnić zasoby pamięci przed wyjściem z programu. Destruktory nie mogą być dziedziczone ani przeciążone.

```
using System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            KlasaZDestruktorem kzd = new KlasaZDestruktorem();
            kzd.PobierzLiczbe();
            kzd.WyswietlLiczbe();
            Console.ReadKey();
            // Wynik działania programu
            //Obiekt jest tworzony
            //Liczba: 100
            //Obiekt jest kasowany
        }
    }
    class KlasaZDestruktorem
    {
        private int liczba;
        // konstruktor
        public KlasaZDestruktorem()
        {
            Console.WriteLine("Obiekt jest tworzony");
        }
        // destruktor
        ~KlasaZDestruktorem()
        {
            Console.WriteLine("Obiekt jest kasowany");
        }
        public void PobierzLiczbe()
        {
            liczba = 100;
        }
        public void WyswietlLiczbe()
        {
            Console.WriteLine("Liczba: {0}", liczba);
        }
    }
}
```

Statyczne składniki klasy

Składniki statyczne możemy zdefiniować za pomocą słowa kluczowego **static**. Jeżeli zadeklarujemy składową jako statyczną nie ważne jak wiele obiektów klasy utworzymy, istnieje zawsze tylko jedna kopia składowej statycznej.

Słowo kluczowe **static** mówi nam o tym, że istnieje tylko jedna instancja składowej dla klasy. Zmienne statyczne używane są przeważnie do definiowania stałych, ponieważ ich wartości mogą być wywołane bez tworzenia instancji danej klasy. Statyczne zmienne mogą być zainicowane na zewnątrz metody lub definicji klasy. Mogą również zostać zainicowane wewnątrz definicji klasy.

```
using System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            StatyczneSkladowe ss1 = new StatyczneSkladowe();
            ss1.Dodaj();
            ss1.Dodaj();
            ss1.Dodaj();
            // Teraz utworzymy kolejny obiekt typu StatyczneSkladowe
            StatyczneSkladowe ss2 = new StatyczneSkladowe();
            // Pamiętajcie, że mamy tylko jedną kopię zmiennej statycznej
            ss2.Dodaj();
            ss2.Dodaj();
            ss2.Dodaj();
            Console.WriteLine("Wartość liczby z pierwszego obiektu: {0}", ss1.WyswietlNumer
());
            Console.WriteLine("Wartość liczby z drugiego obiektu: {0}", ss2.WyswietlNumer()
);
            Console.ReadKey();
            // Jeżeli wynik jest dla Ciebie zaskakujący przeczytaj jeszcze raz
            // definicję składowych statycznych
            // Wynik działania programu
            //Wartosc liczby z pierwszego obiektu: 6
            //Wartosc liczby z drugiego obiektu: 6
        }
    }
    class StatyczneSkladowe
    {
        public static int liczba;
        public void Dodaj()
        {
            liczba++;
        }
        public int WyswietlNumer()
        {
            return liczba;
        }
    }
}
```

Można również zdefiniować metody klasy jako statyczne. Takie metody mają dostęp tylko do zmiennych statycznych. Funkcje statyczne istnieją nawet zanim obiekt zostanie utworzony. Poniżej przykład użycia statycznych metod:

```
using System;
namespace Klasy
{
    class Program
    {
        static void Main(string[] args)
        {
            StatycznaMetoda sm = new StatycznaMetoda();
            sm.Dodaj();
            sm.Dodaj();
            sm.Dodaj();
            // Metoda statyczna jest dostępna bez tworzenia obiektu klasy
            Console.WriteLine("Wartość liczby: {0}", StatycznaMetoda.ZwrocLiczbe());
            Console.ReadKey();
            // Wynik działania programu
            //Wartosc liczby: 3
        }
    }
    class StatycznaMetoda
    {
        public static int liczba;
        public int liczba2;
        public void Dodaj()
        {
            liczba++;
        }
        public static int ZwrocLiczbe()
        {
            // w metodzie nie mamy dostępu do zmiennej liczba2
            return liczba;
        }
    }
}
```

Klasa statyczna

Warto również wiedzieć, że możemy utworzyć statyczną klasę. Klasa taka mówi, iż nie została napisana po to, aby tworzyć nowe obiekty. Nawet, gdybyśmy chcieli utworzyć nowy obiekt klasy statycznej kompilator zgłosi błąd.

W klasie takiej mogą znajdować się tylko statyczne składowe. Prostym przykładem takie klasy może być klasa odpowiedzialna za wykonywanie obliczeń matematycznych. Nie chcemy tworzyć instancji klasy za każdym razem, kiedy chcemy wykonać dodawanie lub odejmowanie pomiędzy liczbami. W takim przypadku warto przygotować klasę statyczną:

```
static class OperacjeMatematyczne
{
    static int DodajLiczba(int a, int b)
    {
        return a + b;
    }
    static int OdejmijLiczby(int a, int b)
    {
        return a - b;
    }
}
```

Zadania do zrobienia

1. Utwórz klasę **Animal** która będzie miała konstruktor:
    a. Bezparametrowy który ustawi 3 parametry na ustawienai domyśle:
        i. Name – unknown – (string)
        ii. Weight – 0 (int)
        iii. Height – 0 (int)
2. Utwórz klasę **Animal** która będzie miała konstruktor:
    a. Który przyjmie parametry i ustawi 3 parametry z zadania 1 na określone wartości podane przez użytkownika dla 4 zwierząt:
        i. Tygrys 300 120
        ii. Slon 4500 350
        iii. Kapibara 35 30
        iv. Kon polski 400 180
3. Utwórz klase **Car**, która będzie reprezentowała samochody z podanej marki (**Brand**), modelu(**Model**) i roku produkcji(**Year**). Utwórz konstruktor bezparametrowy. Wprowadź **5** samochodów podanych poniżej oraz **2** wybrane przez Ciebie spoza listy:
        a. ('Ford', 'Mustang', '1964'),
        b. ('Volvo', 'p1800', '1968'),
        c. ('BMW', 'M1', '1978'),
        d. ('Toyota', 'Celica', '1975'),
        e. ('Porche', '911', '1965');

    Wyświetl wszystkie wprowadzone samochody.