Laboratorium 4

Programowanie w C# (semestr zimowy 2019/2020)

Temat: Metody

Prowadzący: Piotr Pięta

1. Wprowadzenie

Metody (inaczej: funkcje) pozwalają programiście opisać część rzeczywistości. W *paradygmacie obiektowym* (o nim, szerzej – na kolejnych laboratoriach) program składa się z klas (nowy typ danych stworzony przez programistę) oraz obiekty (ukonkretnione klasy – instancje klas, „na nich bazujemy"). Obiektowi takiemu możemy najczęściej przypisać nie tylko dane, np. pojemność silnika, ilość kół itp., ale również dokonać pewnej próby deskrypcji jego mechanizmów umożliwiających zmianę wewnętrznych danych, a w konsekwencji stanu obiektu, czyli opisać zachowanie. Ta próba zamodelowania zachowania obiektu jest szczególnie interesująca (i bliska). Odbywa się ona m.in. za pomocą metod. Oczywiście nie każde zachowanie obiektu jest uzasadnione, np. obiekt samochód i funkcja *latanie()* nie brzmi najlepiej, nie spodziewamy się takiej funkcjonalności samochodu, ale *zwolnij()*, *ominPatrol()*, *dodajGazu*() już tak [przykład mniej formalny].

```
class Program0
{
	static void Main()
	{
		string firstName;
		string lastName;

		Console.WriteLine("Hej, ty!");
		Console.Write("Wprowadź imię: ");
		firstName =Console.ReadLine();

		Console.Write("Wprowadź nazwisko: ");
		lastName =Console.ReadLine();
		Console.WriteLine(
		$"Twoje imię i nazwisko to { firstName } {
		lastName }.");
	}}
```

Z gotowych metod korzystaliście Państwo wielokrotnie, nie będąc zawsze świadomymi. Pierwsza metoda, która „przewija się" wraz z

wzrastającą znajomością C# to metoda *Main()*, która…. No właśnie, co robi? ☺

Metoda statyczna (słowo kluczowe **static**) – metoda, która nie jest wywoływana dla jakiegokolwiek obiektu danej klasy.

Język C# nie obsługuje metod globalnych. Wszystkie metody muszą się znajdować w deklaracjach typów. To dlatego metodę Main() oznaczono jako statyczną (static). Jest to odpowiednik metod globalnych z języka C++ i metod shared z języka Visual Basic.

Przykład: metoda klasy Math: Math.Sin(x)

1.1 Definicja metody

```
[Modyfikatory] Typ NazwaMetody ([Lista argumentów])
{
      [Ciało metody]
      //…;
      //…;
}
```

1.1.1
Modyfikatory - określają zachowanie i dostępność danej metody

public udostępnia daną metodę na zewnątrz dla innych klas
private umożliwia użycie metody tylko wewnątrz klasy

Modyfikator *private* jest domyślny dla składowych klas.

Typ – typ (zwracany) danej zwracanej przez metodę, np. int
NazwaMetody - nazwa metody, różna od nazwy klasy, w której została zdefiniowana
Lista argumentów – zawiera dane wejściowe metody
Ciało metody – instrukcje wewnątrz bloku funkcji

Powstałą funkcję (zdefiniowaną) można następnie **użyć**, czyli ją **wywołać** w programie (wykorzystując do tego nazwę nowopowstałej funkcji, wraz z parametrami – jeżeli są – umieszczonymi w nawiasie ()).

Pierwsza linia definicji, która obejmuje *modyfikatory, typ, nazwę i listę argumentów* stanowi **deklarację** metody. Deklaracja metody wraz z ciałem metody – stanowią **definicję** metody.

1.2 Prosty przykład – funkcja znajdująca maksymalną wartość

```
public int ZnajdzMax(int a, int b)
 {
       int wynik;
```

```
        if (a > b) //wybór
            wynik = a;
        else
            wynik = b;
        return wynik; //słowo kluczowe return
    }
```

Ciało metody, inaczej treść metody – to kod ujęty w klamrowe nawiasy, który realizuje zadanie przypisane dla metody. W ciele metody można deklarować **zmienne lokalne, umieszczać instrukcje i wywołania innych metod**. Deklarowane zmienne nie mogą mieć tej samej nazwy, co nazwy argumentów.

Jeżeli metoda ma określony typ zwracanej wartości (inny niż *void*), w ciele metody musi być instrukcja (instrukcje) *return*, która zwraca daną określonego typu.

```
class Program
{
    //pierwsza funkcja
    static void Odejmij(int x, int y)
    {
        Console.WriteLine(x - y);
    }

    //kolejna funkcja – Main()
    static void Main(string[] args)
    {
        Odejmij(4, 3); // wywołanie metody
        Console.ReadKey();
    }
}


/************************************************/

class Program2
{
    static double Dziel(double x, int y)
    {
        return (x / y); //DZIELENIE, mianownik!=0
    }
```

```
        static void Main(string[] args)
        {
            Console.WriteLine(Dziel(1.5, 3));
            Console.ReadKey();
        }
    }

        /************************************************/

        static double Dziel(double x, int y)
        {
            double wynik = 0;
            if (y != 0)
            {
                wynik = x / y;
            }
            return (wynik); //gdy warunek
            //niespełniony, wówczas wynik = 0!
        }
```

1.3 Przekazywanie parametrów do funkcji

1.3.1 poprzez wartość (kopia obiektu)

1.3.2 poprzez referencję (adres argumentu – funkcja bazuje na oryginalnych danych – po wywołaniu metody zmiany są widoczne na zewnątrz)

W przypadku typów wbudowanych, aby przekazać argumenty przez referencję należy oznaczyć je przy pomocy jednego z modyfikatorów: **ref** lub **out**. Wybór odpowiedniego modyfikatora zależy od tego, gdzie będzie inicjalizowany argument, jeśli w miejscu wywołania metody, to ref, jeśli wewnątrz metody, to out.

```
        class Program3
        {
            static void Dodaj(ref int a)
            {
                a++;
                Console.WriteLine("Argument z
                wnętrza metody: " + a);
            }
            static void Main(string[] args)
            {
                int x = 5;
```

```
                Console.WriteLine("Przed wywołaniem
                metody: " + x);
                Dodaj(ref x);
                Console.WriteLine("Po wywołaniu
                metody: " + x);
                Console.ReadKey();
            }
        }

        /*****************************************/

        class Program
        {
            static void Dodaj(out int x, out int y)
            {
                x = 2;
                y = 5;
                Console.WriteLine("Dodaj(): x={0}
                y={1}", x, y);
            }

            static void Main(string[] args)
            {
                int a, b; // DEKLARACJA
                Dodaj(out a, out b);
                Console.WriteLine("Main(): a={0}
                b={1}", a, b);
                Console.ReadKey();
            }
        }
```

Argumenty z modyfikatorem *out* to tzw. argumenty wyjściowe, nie można ich przekazać do metody. Z kolei argumenty z modyfikatorem ref to tzn. argumenty wejścia/wyjścia (są przekazywane: do metody/z metody, „w obu kierunkach").

Przekazywanie argumentów przez referencję (*ref* lub *out*) jest jednym ze sposobów uzyskania efektu „zwracania" przez metodę więcej niż jednej wartości wynikowych.

Użycie słowa kluczowego **params** – lista argumentów wejściowych o zmiennej liczbie argumentów

```
class Program4
{
    static void Wielkie(string[] tab)
    {
        for (int i = 0; i < tab.Length; i++)
        {
            tab[i] = tab[i].ToUpper();
```

```
        }
    }

    static void Main(string[] args)
    {
        string[] tab1 = { "jeden", "dwa", "trzy" };
        Wielkie(tab1);
        for (int i = 0; i < tab1.Length; i++)
        {
            Console.Write(tab1[i] + " ");
        }
            Console.ReadKey();
        }
    }}
```

**/*
Metody z ciałem w postaci wyrażenia (C# 6.0)**

```
static string GetFullName( string firstName, string lastName) => $"{ firstName }
{ lastName }";
```

***/**

2. **Zadania do samodzielnego rozwiązania**

2.1 Napisz program zawierający metodę statyczną obliczającą temperaturę w stopniach Fahrenheita na temperaturę w stopniach Celsjusza. Metoda ma przyjmować jeden argument (temperaturę w stopniach Fahrenheita) i zwracać temperaturę w stopniach Celsjusza

2.2. Napisz program wykorzystujący funkcję – program ma pobrać dowolną liczbę całkowitą, a następnie obliczyć sumę cyfr składających się na tę liczbę, np. 45 → na wyjście: 9 (4+5)

2.3 Napisz program wczytujący 3 liczby rzeczywiste a, b, x, a następnie wywołujący metodę, która sprawdza, czy liczba x należy do przedziału obustronnie otwartego (a, b). Metoda sprawdzająca ma zwrócić wartość logiczną, którą należy zinterpretować w metodzie *Main*() z podaniem stosownego komunikatu

2.4 Napisz program, który mnoży elementy tablicy jednowymiarowej przez zadaną liczbę. Mnożenie ma być wykonane w metodzie statycznej przyjmującej jako argumenty tablicę typu *int* oraz liczbę całkowitą (mnożnik).

2.5 Napisz program obliczający potęgę – w osobnej funkcji.

2.6 Napisz program obliczający silnię – w osobnej funkcji.

2.7 Praca własna z dokumentacją – uruchom wybrane przykłady