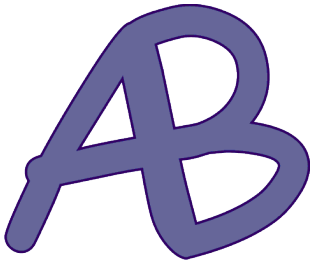


Wykład 3
Składnia języka C# (cz. 2)

Metody

Metody

- ✓ W C# nie jest możliwe definiowanie funkcji niebędących metodami jakiejś klasy.
- ✓ Funkcja może być statyczną składową klasy, ale zawsze jest metodą (tj. właśnie funkcją składową zdefiniowaną w obrębie klasy).

  Metody statyczne, to takie, które można wywołać bez tworzenia instancji klasy, w której są zdefiniowane. Do ich definicji dodamy modyfikator **static**
- ✓ Metody definiować możemy w obrębie istniejącej klasy (klasy Program). Będą to metody statyczne, bo metoda main( ) z której będą one wywoływane jest statyczna.

Metody

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication4
{
    class Program
    {
        static void Metoda()
        {
            Console.WriteLine("Hello World!"); // ciało metody
        }

        static void Main(string[] args)
        {
            Metoda();
        }
    }
}
```

Przeciążanie metod

Język C# umożliwia definiowanie wielu metod o tych samych nazwach, pod warunkiem że różnią się parametrami (dzięki temu mają również inne sygnatury). Nazywa się to przeciążaniem metody (ang. overload). Niemożliwe jest natomiast definiowanie dwóch metod różniących się jedynie zwracanymi wartościami.

```
static void Metoda()
{
    Console.WriteLine("Hello World!");
}
static void Metoda(string tekst)
{
    Console.WriteLine(tekst);
}
static void Main(string[] args)
{
    Metoda();
    Metoda("Witaj, świecie!");
}
```

Domyślne wartości metod

Możliwe jest ustalanie domyślnych wartości parametrów metod. Dzięki temu przy wywołaniu metody argument jest opcjonalny — jeżeli nie wystąpi w liście argumentów w instrukcji wywołania metody przyjmie wartość domyślną

```
static void Main(string[] args)
{
    Przywitaj_sie();          //wypisze Witaj
    Przywitaj_sie("Cześć");   //wypisze Cześć
    Console.ReadKey();
}

private static void Przywitaj_sie(string s = "Witaj")
{
    Console.WriteLine(s);
}
```

Domyślne wartości metod

PRZYKŁAD

```
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication4
{
    class Program
    {
        static void Metoda(string tekst, ConsoleColor kolor = ConsoleColor.White)
        {
            ConsoleColor bieżącyKolor = Console.ForegroundColor;
            Console.ForegroundColor = kolor;
            Console.WriteLine(tekst);
            Console.ForegroundColor = bieżącyKolor;
        }
        static void Main(string[] args)
        {
            Metoda("Witaj Świecie", ConsoleColor.Cyan);
            Metoda("Witaj Świecie");
            Console.ReadKey();
        }
    }
}
```

Argumenty nazwane

Możliwa jest identyfikacja parametrów nie za pomocą ich kolejności, a przy użyciu ich nazw, np.:

Dla metody:

```
static void Metoda(string tekst, ConsoleColor kolor = ConsoleColor.White)
{…..}
```

Poprawne są oba wywołania

```
Metoda(kolor: ConsoleColor.Green, tekst: "Witaj, świecie!");
Metoda(tekst: "Witaj, świecie!", kolor: ConsoleColor.Green);
```

Zaletą tego rozwiązania jest czytelność kodu.

Przekazywanie argumentów do metody przez wartości i referencje

```
static private void zwiększ(int liczba)
{
    liczba = liczba + 100;
    Console.WriteLine("Po zwiekszeniu: " + liczba);
}
static void Main(string[] args)
{
    int x = 0;
    zwiększ(x);
    Console.WriteLine("Po wyjsciu z metody: " + x);
    Console.ReadKey();
}
```

```
Po zwiekszeniu: 100
Po wyjsciu z metody: 0
```

```
Po zwiekszeniu: 100
Po wyjsciu z metody: 100
```

```
static private void zwiększ(ref int liczba)
{
    liczba = liczba + 100;
    Console.WriteLine("Po zwiekszeniu: " + liczba);
}
static void Main(string[] args)
{
    int x = 0;
    zwiększ(ref x);
    Console.WriteLine("Po wyjsciu z metody: " + x);
    Console.ReadKey();
}
```

Zmienne wskaźnikowe i dynamiczne

Wskaźniki

Zmienne wskaźnikowe przechowują adresy do zmiennych prostych i złożonych (struktur, tablic, obiektów itp.)

Przykład	**Opis**
int* p	p to wskaźnik do liczby całkowitej.
int** p	p to wskaźnik do wskaźnika do liczby całkowitej.
int*[ ] p	p to tablica wskaźników do liczb całkowitych.
char* p	p jest wskaźnik do znaku.
void* p	p to wskaźnik do nieznanego typu.

Wskaźniki

Operacje na wskaźnikach

Operator	**Zastosowanie**
*	wykonuje operację wskaźnika pośredniego.
->	dostęp do elementu struktury za pomocą wskaźnika.
[ ]	indeksuje wskaźnik.
&	uzyskuje adres zmiennej.
++ oraz --	zwiększa i zmniejsza wartość wskaźnika.
+ oraz -	wykonuje operacje arytmetyczne na wskaźniku.

Zmienne dynamiczne

Zmienne proste możemy deklarować jako statyczne lub dynamiczne (na stosie lub stercie) – w C# nie ma to jednak tak dużego znaczenia jak w C++

```
int liczba1;
int liczba2 = new int( );

char znak1;
char znak2 = new char( );
```

```
int32 liczba = new Int32();
int liczba = new int();
int liczba = 1;
```

Kolekcje

Pojęcie kolekcji

- ✓ Struktury danych: tablice, listy, kolejki, drzewa itp. zostały w C# nazwane kolekcjami.
- ✓ Typowe kolekcje zostały zaimplementowane w platformie .NET i są gotowe do użycia.
- ✓ Kolekcje zawarte są przestrzeni nazw System.Collections.Generic
- ✓ oraz System.Collections.Specialized zawierającej wyspecjalizowane wersje kolekcji.

Tablice

- Tablica jest w instancją klasy System.Array
- Składnia deklaracji referencji (wskaźnika) do tablicy elementów typu int:

```
int[ ] tab;
```

- Deklaracji referencji wraz z utworzeniem obiektu tablicy (rezerwowana jest pamięć na stercie):

```
int[ ] tab = new int[100];
```

Po utworzeniu obiektu tablicy jest ona automatycznie inicjowana wartościami domyślnymi dla danego typu czyli w przypadku typu int — zerami.

- Inicjalizacja elementów tablicy:

```
int[ ] tab = new int[ 3 ] { 1 , 2 , 4 };
```

```
Int[ ] tab = new int[ ] {0,1,2,3,4,5,6,7,8,9}; //tablica 10-cio elementowa
```

Tablice wielowymiarowe

- Deklaracji referencji wraz z utworzeniem obiektu tablicy dwuwymiarowej

```
int[ , ] tab2D = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };
```

Przykład:

```
int[,] tab2D = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 3; j++)
        Console.Write(tab2D[i, j]);
    Console.WriteLine();
}
```

Pętla foreach

Instrukcja foteach wykonuje instrukcję lub blok instrukcji dla każdego elementu w określonym wystąpieniu typu, który implementuje (np. tablicy)

```
int[] tab = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
foreach (int i in tab)
{
    System.Console.Write(i+" ");
}
```

Instrukcja foteach działa także na tablicach wielowymiarowych

```
int[,] tab2D = new int[3, 2] { { 100, 200 }, { 101, 201 }, { 102, 202 } };
// int[,] tab2D = { { 100, 200 }, { 101, 201 }, { 102, 202 } };
foreach (int i in tab2D)
{
    System.Console.Write(i+" ");
}
```

Tablice – metody klasy System.Array

Niezwykle przydatną operacją na tablicach jest sortowania

```
int[] tab = new int[50];
Random r = new Random();
for (int indeks = 0; indeks < tab.Length; indeks++)
    tab[indeks] = r.Next(100);

string s = "Wylosowana tablica: ";
foreach (int los in tab) s += los.ToString() + " ";
Console.WriteLine(s);

Array.Sort(tab);  //całe sortowanie w jednej linijce ;)

s = "Tablica po posortowaniu: ";
foreach (int los in tab) s += los.ToString() + " ";
Console.WriteLine(s);
```

Tablice – metody klasy System.Array

Inne przydatne matody klasy System.Array

Właściwość Length – zwraca długość tablicy.

```
tab.Length
```

Właściwość Rank – zwraca liczbę wymiarów tablicy

```
tab.Rank
```

Metoda Initialize( ) – inicjuje table wartościociami 0, null, false (zależnie od typu elementów)

```
tab.Initialize()
```

Tablice – metody klasy System.Array

Metoda Sort( ) – sortuje tablicę

```
Array.Sort(tablica);
```

Metoda Resize( ) – zmienia rozmiar tablicy

```
Array.Resize( ref tablica, tablica.Lenght + 5)
```

Metoda Clear( ) – zeruje określoną liczbę elementów tablicy (parametry: tablica, indeks początkowy, indeks końcowy)

```
Array.Clear (tablica, 0, tabliaca.Length -1)
```

Metoda Clone( ) – tworzy kopię tablicy

```
Int[ ] kopia = (int[ ]) tablica.Clone( );
```

Metoda Copy( ) – Kopiuje elementy do wskazanej tablicy (parametry: tablica źródłowa, tablica docelowa, liczba elementów)

```
Array.Copy (tablica, tablica_docelowa, tabliaca.Length)
```

Tablice – metody klasy System.Array

Metoda IndexOf( ) – zwraca index elementu spełniającego podane kryteria

```
int pozycja = Array.IndexOf(tablica, szukan_wartosc);
```

Metoda BinarySearch() – wyszukiwanie binarne w tablicy – zwraca numer indeksu na którym występuje szukana wartość, lub -1 w przypadku braku dopasowań (Uwaga – stosujemy do tablic posortowanych)

```
int pozycja = Array.BinarySearch(tablica, szukan_wartosc);
```

Tablice – metody klasy System.Array

Metoda Find( ) – wyszukuje pierwszy element spełniający podane kryteria

```
int pozycja = Array.Find(tablica, funkcja_testująca);
```

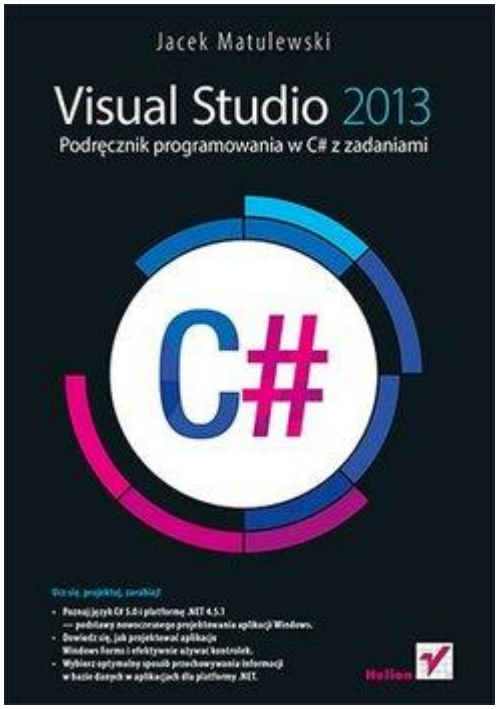
Metoda FindAll( ) – wyszukuje wszystkie elementy spełniający podane kryteria. Wynik zwraca w postaci tablicy

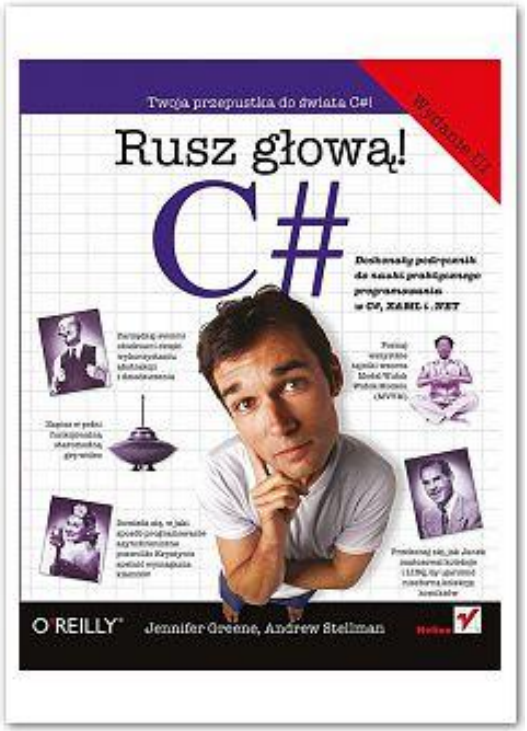
```
int[ ] pozycje = Array.FindAll(tablica, funkcja_testująca);
```

Obie powyższe metody wymagają zdefiniowania funkcji testującej, która otrzymuje w parametrze wartość z tablicy i zwraca wartość prawda/fałsz

```
static bool funkcja_testujaca(int obj)
{
    if (obj % 3 == 0) return true; else return false;
}
```

Literatura:





Użyte w tej prezentacji tabelki pochodzą z książki:Visual Studio 2013. Podręcznik programowania w C# z zadaniami
Autor: Matulewski Jacek, Helion