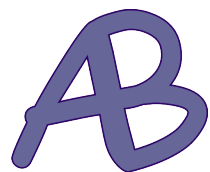
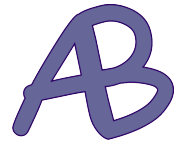


Wykład **Składnia języka C#** (cz. 2) **Kolekcje**

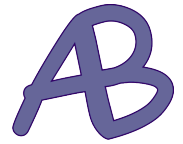


Kolekcje



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 w 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 sterpie):

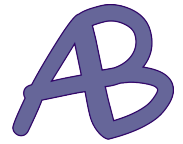
```
int[ ] tab = new int[100];
```

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
```



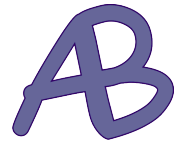
Tablice wielowymiarowe

- 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 **foreach** 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 **foreach** 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

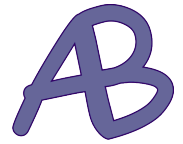
Niezwykle przydatną operacją na tablicach jest sortowania

```
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 linii ;)

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



Tablice – metody klasy System.Array

Inne przydatne metody klasy System.Array

Właściwość Length – zwraca długość tablicy.

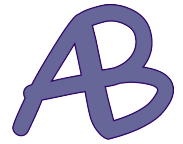
`tab.Length`

Właściwość Rank – zwraca liczbę wymiarów tablicy

`tab.Rank`

Metoda Initialize() – inicjuje tabele wartościami 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.Length + 5)`

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

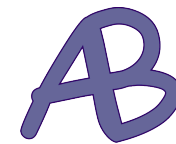
`Array.Clear (tablica, 0, tablica.Length - 1)`

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, tablica.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 wyłącznie do tablic posortowanych)

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

Tablice – metody klasy System.Array

Metoda Find() – wyszukuje pierwszy element spełniający podane kryteria

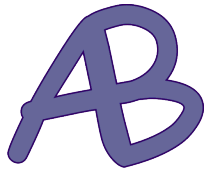
```
int pozycja = Array.Find(tablica, funkcja_testująca);
```

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

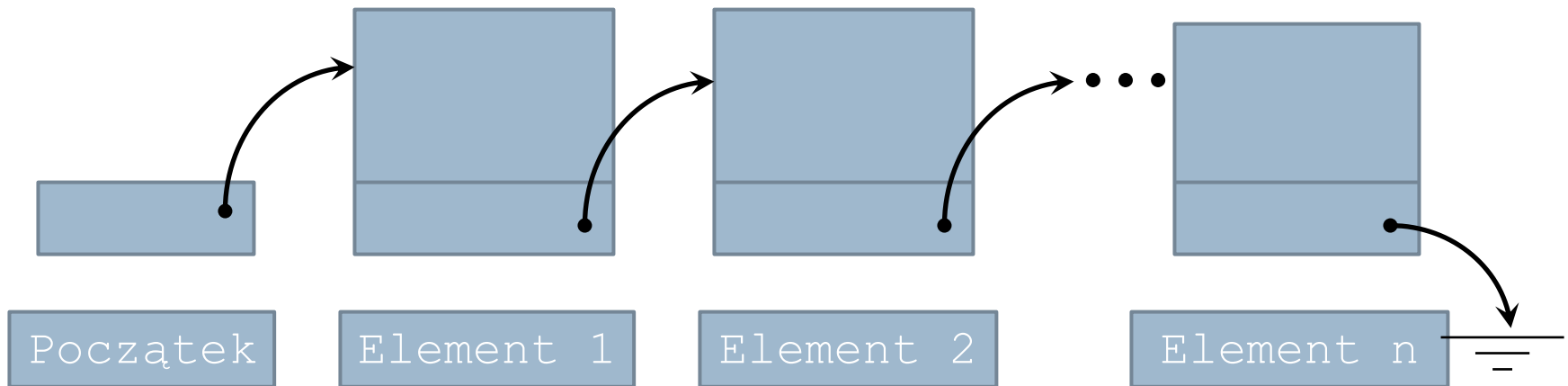
```
int[ ] pozycje = Array.FindAll(tablica, funkcja_testująca);
```

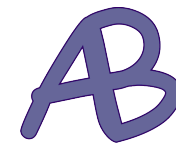
Obie powyższe metody wymagają zdefiniowania **funkcji testującej**, która otrzymuje w parametrze wartość z tablicy i zwraca wartość prawda/fałsz

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



Kolekcje „List” i „SortedList”





Kolekcja „Listy”

Lista - należy do grupy typów ogólnych (ang. generic types).

- ✓ W porównaniu z tablicą (Array) ma tą zaletę, że liczba elementów może być zmieniana już po utworzeniu listy.
- ✓ Można dodawać elementy na koniec, na początek i w środek listy.
- ✓ Można usuwać dowolny element listy.
- ✓ Dostęp do dowolnego elementu listy możliwy jest, tak samo jak w przypadku tablicy - za pomocą operatora **[]**



Kolekcja „Listy”

Tworzenie listy:

```
List<typ> l = new List<typ>(tab. wart. inicjalizujących);
```

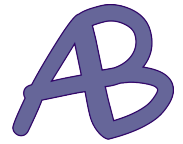
```
List<int> l = new List<int>();
```

```
List<String> s = new List<String>();
```

W parametrze konstruktora listy możemy podać tablicę wartości inicjalizujących.

```
List<int> l = new List<int>(new int[] {1,2,3,4,5});
```

```
List<String> s = new List<String>(new String[] {"aa","bb","cc"});
```



Kolekcja „Listy”

Podstawowe operacje na listach (na przykładzie listy zawierającej elementy typu String):

```
List<String> nazwa = new List<String>();
```

```
nazwa.Add("element");
```

- Dodawanie elementu

```
nazwa.AddRange(new String[] { "aa", "bb" });
```

- Dodanie tablicy elementów (na koniec listy)

```
nazwa.Insert(poz, "aa");
```

- wstawianie elementu na wskazaną pozycję **UWAGA:** nie zastępujemy tylko wstawiamy

```
nazwa.InsertRange(poz, new String[] { "aa", "bb" });
```

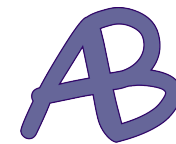
- wstawianie listy elementów na wskazaną pozycję

```
nazwa.RemoveAt(poz);
```

- usunięcie wskazanego elementu (o wsk. indeksie)

```
nazwa.Remove("bb");
```

- usunięcie elementu o wskazanej wartości,



Kolekcja „Listy”

Podstawowe operacje na listach c.d.:

`nazwa.Clear();`

- wyczyszczenie listy

`nazwa.Sort();`

- sortowanie listy

`nazwa.Reverse();`

- odwrócenie listy

`nazwa.Count();`

- podaje liczbę elementów

`nazwa.ToArray(TablicaDocelowa);`

- eksportuje listę do tablicy.


```
class Program
```

```
{
```

```
    static Random r = new Random();
```

```
    static List<int> l = new List<int>(new int[] { 1, 2, 3, 4, 5 });
```

```
    static void Main(string[] args)
```

```
{
```

```
        wypisz("test 1");
```

```
        for (int i = 0; i < 10; i++)
```

```
            l.Add(i);
```

```
        wypisz("test 2");
```

```
        l.InsertRange(0, new int[] { 10, 20, 30, 40, 50 });
```

```
        wypisz("test 3");
```

```
        l.Insert(0, 100);
```

```
        wypisz("test 4");
```

```
        for (int i = 0; i < l.Count; i++)
```

```
            if (l[i] % 5 == 0) { l.RemoveAt(i); i--; }
```

```
        wypisz("test 5");
```

```
        Console.ReadKey();
```

```
    }
```

```
    static void wypisz(String opis="Zawartosc")
```

```
{
```

```
        String s = opis+": ";
```

```
        for (int i = 0; i < l.Count; i++)
```

```
            s += l[i].ToString() + " ";
```

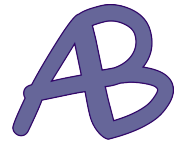
```
        Console.WriteLine(s);
```

```
    }
```

```
}
```



Operacje na liście - przykład



Kolekcja „SortedList”

SortedList - w odróżnieniu od omówionej wcześniej jest „dwukolumnowa”.

- ✓ Każdy element listy przechowuje klucz i wartość (właściwości Key i Value).
- ✓ Pozwala to sortowanie obu wartości według klucza.

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        SortedList<string, string> artysci = new SortedList<string, string>();
```

```
        artysci.Add("Sting", "Gordon Matthew Sumner");
```

```
        artysci.Add("Bolesław Prus", "Aleksander Głowacki");
```

```
        artysci.Add("Pola Negri", "Barbara Apolonija Chałupiec");
```

```
        artysci.Add("John Wayne", "Marion Michael Morrison");
```

```
        artysci.Add("Chico", "Leonard Marx");
```

```
        artysci.Add("Harpo", "Arthur Marx");
```

```
        artysci.Add("Groucho", "Julius Marx");
```

```
        artysci.Add("Bono", "Paul Hewson");
```

```
        artysci.Add("Ronaldo", "Luiz Nazario de Lima");
```

```
        artysci.Add("Madonna", "Madonna Louise Veronica Ciccone");
```

```
        artysci.Add("Gabriela Zapolska", "Maria G. Śnieżko-Błocka");
```

```
        string komunikat = "Zawartość listy:\n";
```

```
        foreach (KeyValuePair<string, string> artysta in artysci)
```

```
            komunikat += artysta.Key + " - " + artysta.Value + "\n";
```

```
        Console.WriteLine(komunikat);
```

```
        Console.ReadKey();
```

```
    }
```

```
}
```

Zawartość listy:

Bolesław Prus - Aleksander Głowacki

Bono - Paul Hewson

Chico - Leonard Marx

Gabriela Zapolska - Maria G. Śnieżko-Błocka

Groucho - Julius Marx

Harpo - Arthur Marx

John Wayne - Marion Michael Morrison

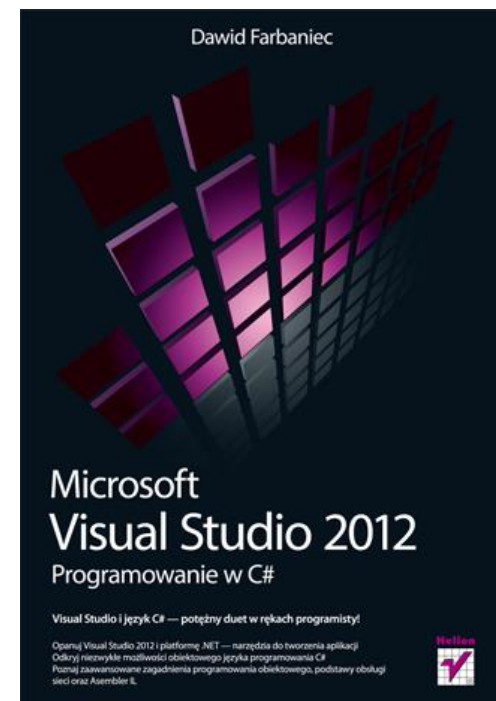
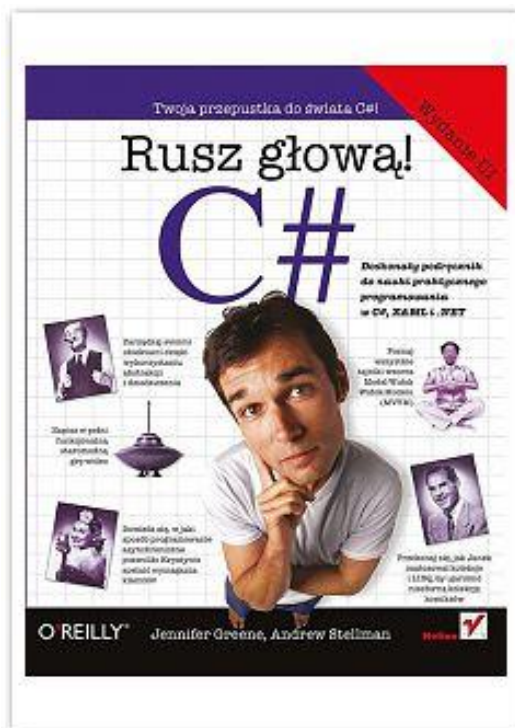
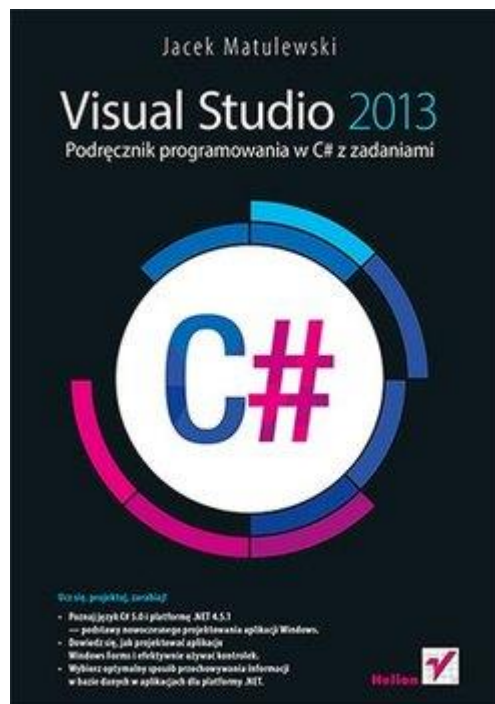
Madonna - Madonna Louise Veronica Ciccone

Pola Negri - Barbara Apolonija Chałupiec

Ronaldo - Luiz Nazario de Lima

Sting - Gordon Matthew Sumner

Literatura:



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