



Operacje na kolekcjach

Bartosz Litwiniuk



Bartosz Litwiniuk

Bartosz.Litwiniuk@outlook.com

Koło naukowe .NET WE,

Komputronik Biznes

Politechnika Poznańska WE

Agenda

Tablice, listy,
słowniki, czyli kilka
słów o kolekcjach

Przeglądanie kolekcji

LINQ

Serializacja kolekcji

Dlaczego warto używać
kolekcji?

Cel wykorzystywania kolekcji

```
//Przykład kodu źródłowego bez wykorzystania kolekcji
```

```
string kolega1 = "Tomasz";  
string kolega2 = "Kamil";  
string kolega3 = "Grzegorz";  
string kolega4 = "Konrad";  
string kolega5 = "Maciej";
```

```
//Wypisanie imion wszystkich kolegów
```

```
Console.WriteLine(kolega1);  
Console.WriteLine(kolega2);  
Console.WriteLine(kolega3);  
Console.WriteLine(kolega4);  
Console.WriteLine(kolega5);
```

```
//Przykład kodu źródłowego z wykorzystaniem kolekcji
```

```
string[] koledzy = new string[] { "Tomasz", "Kamil", "Grzegorz", "Konrad", "Maciej" };
```

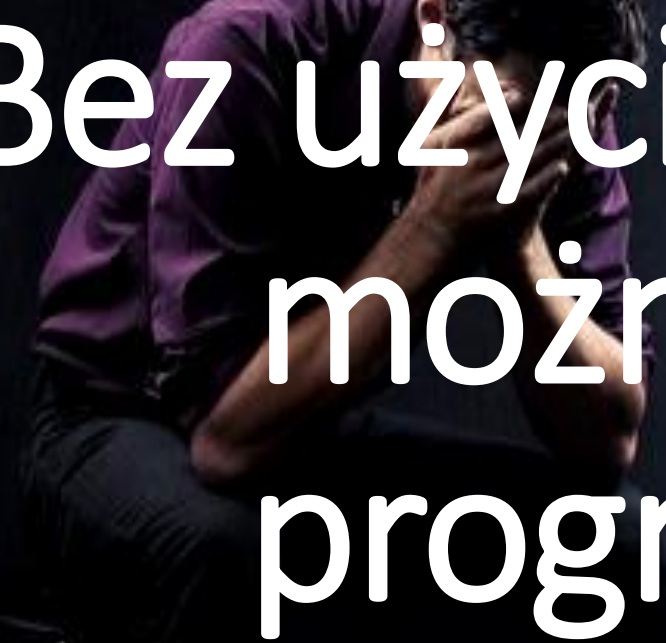
```
foreach (string imieKolegi in koledzy)
```

```
{  
    Console.WriteLine(imieKolegi);  
}
```

```
//Przykład kodu źródłowego z wykorzystaniem kolekcji
```

```
string[] koledzy = new string[] { "Tomasz", "Kamil", "Grzegorz", "Konrad", "Maciej" };
```

```
Array.ForEach(koledzy, x => { Console.WriteLine(x); });
```

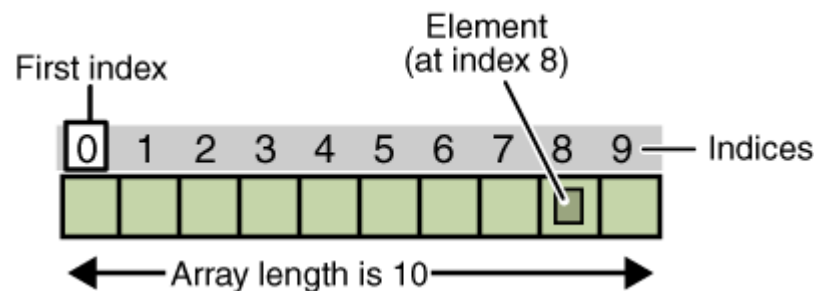


Bez użycia kolekcji nie
można dobrze
programować.

Tablice, listy, kolejki,
czyli kilka słów o
kolekcjach

Tablice, listy, kolejki, czyli kilka słów o kolekcjach

Tablice – kontener uporządkowanych danych tego samego typu

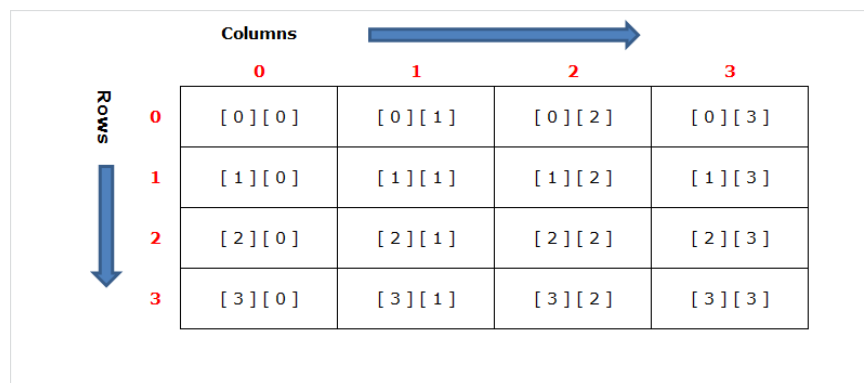


Źródło: <http://www.thecrazyprogrammer.com/wp-content/uploads/2015/05/Array-in-Java.gif>

Przykład tablicy liczb całkowitych o rozmiarze 5 w języku C#
`int[] numbers = new int[5] {1, 2, 3, 4, 5};`

Tablice, listy, kolejki, czyli kilka słów o kolekcjach

Tablice wielowymiarowe



		Columns			
		0	1	2	3
Rows	0	[0][0]	[0][1]	[0][2]	[0][3]
	1	[1][0]	[1][1]	[1][2]	[1][3]
	2	[2][0]	[2][1]	[2][2]	[2][3]
	3	[3][0]	[3][1]	[3][2]	[3][3]

<http://www.advancesharp.com/BlogImages/1047-two-dimensional-array.png>

Przykład w języku C#

```
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

Co powinno się tu znaleźć?

Tablice, listy, kolejki, czyli kilka słów o kolekcjach

Listy – dynamiczna struktura danych, w której elementy umieszczone są w liniowym porządku. Charakterystyczną cechą odróżniającą listę od tablicy jest możliwość dodawania lub usuwania dowolnych elementów. Szczególnym przypadkiem listy jest stos i kolejka.

```
List<string> listaElementow = new List<string>();  
lista.Add("Element 1");  
lista.Add("Element 2");  
lista.Add("Element 3");
```

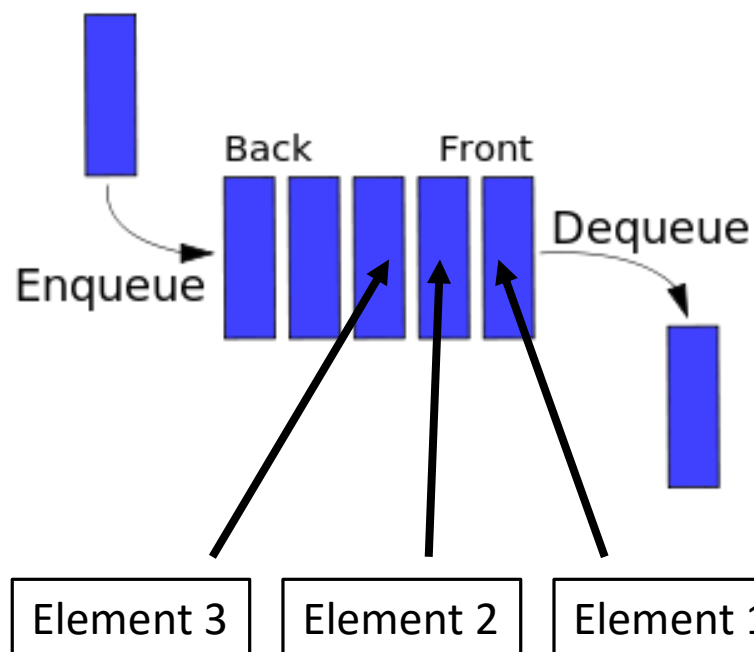
```
List<string> lista = new List<string>();  
lista.Add("Element 1");  
lista.Add("Element 2");  
lista.Add("Ciąg znaków");
```

Index	Value
[0]	"Element 1"
[1]	"Element 2"
[2]	"Ciąg znaków"

Tablice, listy, kolejki, czyli kilka słów o kolekcjach

Kolejka – liniowo uporządkowana struktura danych, dodajemy elementy na koniec kolejki, a usuwamy z początku kolejki

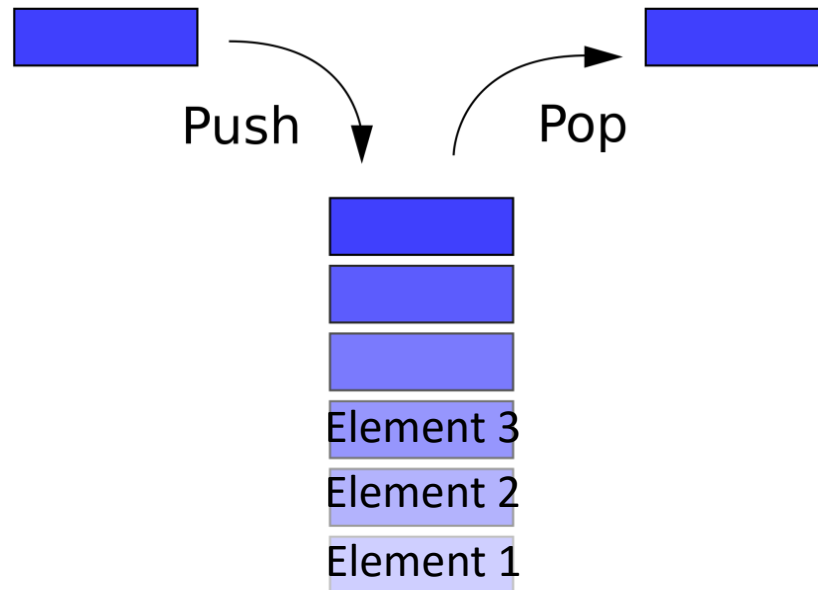
https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/300px-Data_Queue.svg.png



```
Queue<string> kolejka = new Queue<string>();  
  
kolejka.Enqueue("Element 1");  
kolejka.Enqueue("Element 2");  
kolejka.Enqueue("Element 3");  
Console.WriteLine(kolejka.Dequeue()); // ?
```

Stos

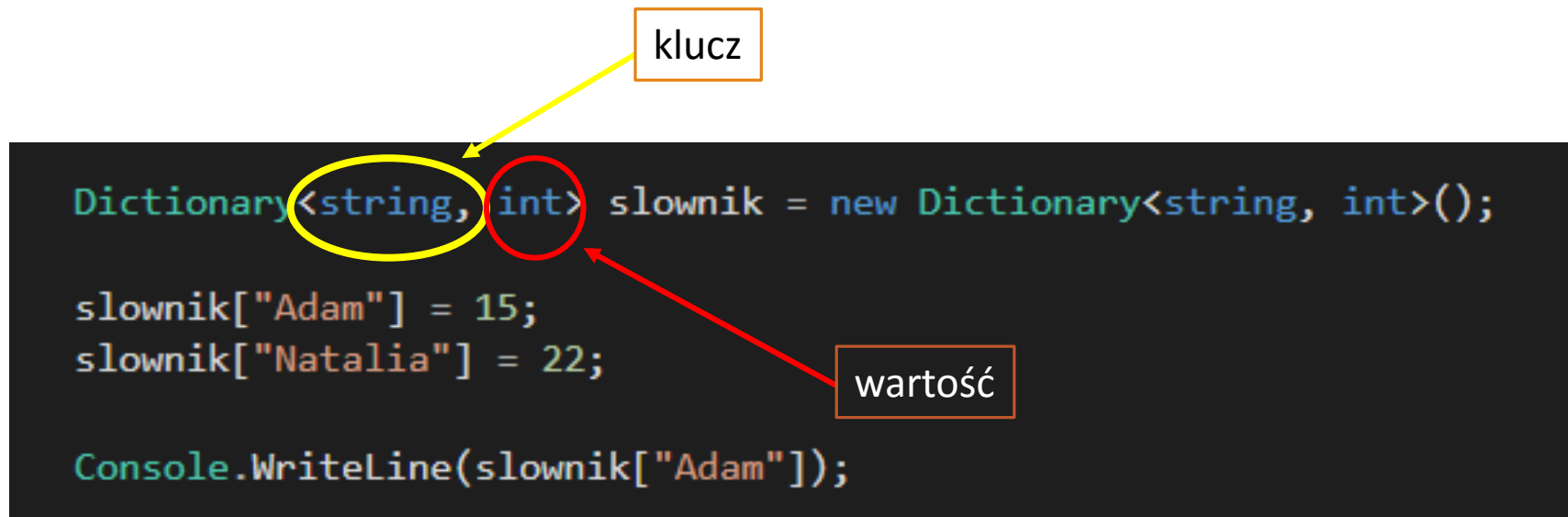
Stos (ang. Stack) – struktura danych, w której dane dokładane są na wierzch stosu i z wierzchołka stosu są pobierane (bufor typu **LIFO**, *Last In, First Out*; *ostatni na wejściu, pierwszy na wyjściu*). Dostęp jedynie do elementu który jest na wierzchu stosu. (Źródło: wikipedia.org)



```
Stack<string> stack = new Stack<string>();  
stack.Push("Element 1");  
stack.Push("Element 2");  
stack.Push("Element 3");  
  
string element = stack.Pop();
```

Słownik

Słownik (ang. Dictionary) – struktura danych przechowująca pary postaci KLUCZ, WARTOŚĆ, pozwalająca na dostęp do wartości poprzez podanie klucza.



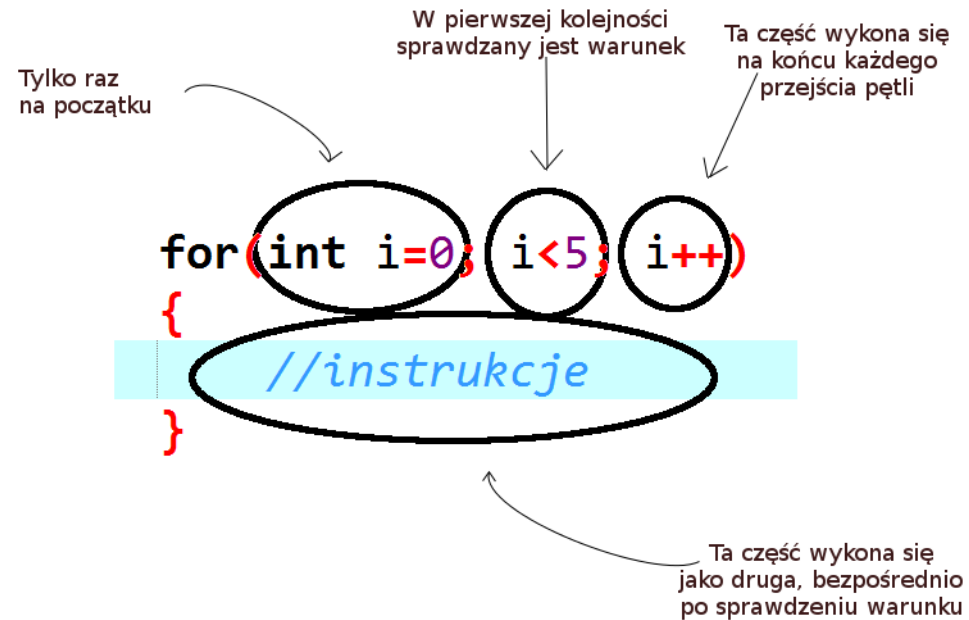
The diagram illustrates the components of a C# Dictionary. A yellow oval highlights the 'string' type in the first line of code, with a yellow arrow pointing to it from a box labeled 'klucz'. A red oval highlights the 'int' type in the same line, with a red arrow pointing to it from a box labeled 'wartość'. The code is as follows:

```
Dictionary<string, int> slownik = new Dictionary<string, int>();  
  
slownik["Adam"] = 15;  
slownik["Natalia"] = 22;  
  
Console.WriteLine(slownik["Adam"]);
```

Sposoby przeglądania kolekcji

Pętla for

Służy do wielokrotnego wykonywania tego samego bloku instrukcji.



Źródło: <http://www.algorytm.edu.pl/images/petlafor.png>

Pętla while

Umożliwia powtarzanie instrukcji dopóki warunek jest spełniony.

Przykład w języku C#

```
int i = 0;  
while(i < 5)  
{  
    Console.WriteLine(i);  
    i++;  
}
```

warunek

instrukcje

Pętla foreach

Służy do iterowania się po kolekcji elementów, pętla automatycznie przy każdej iteracji przypisuje zadanej w nagłówku zmiennej kolejną wartość z kolekcji.

```
List<string> lista = new List<string>();  
lista.Add("Element 1");  
lista.Add("Element 2");  
lista.Add("Element 3");  
  
foreach (string value in lista)  
{  
    Console.WriteLine(value);  
}
```

Przeglądanie kolekcji z wykorzystaniem pętli



```
string[] samochody = new string[] { "Ford Mustang", "Chevrolet Camaro",  
                                     "Corvette c6", "Lamborghini gallardo",  
                                     "Dodge viper", "Porsche spyder" };
```

```
//for  
for (int i = 0; i < samochody.Length; i++)  
{  
    Console.WriteLine(samochody[i]);  
}
```

```
//while  
int j = 0;  
while(j < samochody.Length)  
{  
    Console.WriteLine(samochody[j]);  
    j++;  
}
```

```
//foreach  
foreach(string nazwaSamochodu in samochody)  
{  
    Console.WriteLine(nazwaSamochodu);  
}
```

LINQ

Interfejsy kolekcji

IEnumerable	ICollection	IQueryable
Najbardziej podstawowa kolekcja, tylko do odczytu, nie pozwala na dodawanie, modyfikację i usuwanie elementów. Służy do operowania na już pobranych danych i przechowywanych w pamięci.	Dziedziczy po IEnumerable , pozwala na dodawanie i usuwanie elementów.	Przeznaczone do źródeł zewnętrznych, pobiera jedynie dane spełniające określone warunki.

LINQ

Language Integrated Query - zintegrowane zapytania językowe.

- Pozwala na szybkie tworzenie zapytań do kolekcji.
- Wyrażenia LINQ możemy tworzyć z wykorzystaniem dwóch notacji

```
int[] liczby = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
int[] kolekcjaLiczbyWiększychOd5 = (from liczba in liczby  
                                     where liczba > 5  
                                     select liczba).ToArray();
```

```
int[] posortowanaKolekcja = (from liczba in liczby  
                             orderby liczby  
                             select liczba).ToArray();
```

```
int[] liczbyWiększeOd5 = liczby.Where(x => x > 5).ToArray();
```

```
int max = liczby.Max(x => x);
```

```
int[] posortowanaKolekcja = liczby.OrderBy(x => x).ToArray();
```

LINQ - możliwości

- First() – pierwszy element z kolekcji
- FirstOrDefault() – pierwszy element z kolekcji lub null w przypadku pustej kolekcji
- Last() – ostatni element z kolekcji
- Where(...) – filtrowanie
- OrderBy(...) – szeregowanie
- GroupBy(...) - grupowanie
- Any() – sprawdza czy istnieje jakikolwiek element w kolekcji
- Join(...) – łączenie
- Select(...) - wybieranie

Demo

<http://pastebin.com/vfeNFjSG>



Pytania?

TechDay

Szkolenie Arduino

Szkolenie Raspberry pi

C++ 17

Universal Windows Platform



Dziękuję za uwagę!

Bartosz Litwiniuk

bartosz.litwiniuk@outlook.com

Koło naukowe .NET WE,
Komputronik Biznes
Politechnika Poznańska WE