Tablice
Napisy
Kolekcje

Tablice

- Tablica jest szeregiem danych zawierającym określoną liczbę elementów tego samego typu.
- Elementy tablicy są składowane w jednym ciągłym bloku pamięci, co pozwala na szybki dostęp.
- Dostęp do elementów jest swobodny (tzn. można w danym momencie odczytać i zapisać dowolny element, dowolną ilość razy).

Składnia deklaracji tablicy

typ[] Identyfikator;

- *typ*
  Typ danych elementów tablicy (wymagane);

- *Identyfikator*
  Nazwa tablicy (wymagane).

Przykład:

int[] Liczby;

Tablice cd.1

- Zadeklarowanie zmiennej tablicowej nie tworzy instancji tablicy. Jest to jedynie zmienna referencyjna, która będzie powiązana z fizyczną tablicą dopiero po utworzeniu instancji tej tablicy.

- Bazową klasą dla wszystkich tablic w języku C# jest klasa *SystemArray,* więc w momencie utworzenia tablicy, można korzystać z metod i właściwości tej klasy.

Deklaracja tablic wielowymiarowych

- Tablice mogą być jedno lub wielowymiarowe. W przypadku deklaracji tablic jednowymiarowych, używa się pustych nawiasów kwadratowych:

int[ ] Liczby; // deklaracja tablicy jednowymiarowej

- W przypadku tablic wielowymiarowych w nawiasach kwadratowych należy wstawić znaki przecinka, które oddzielają poszczególne wymiary. Tak więc tablicę dwuwymiarową deklaruje się poprzez umieszczenie pojedynczego przecinka wewnątrz nawiasów, tablicę trójwymiarową poprzez umieszczenie dwóch przecinków, etc:

int [,] LiczbyDw; // deklaracja tablicy dwuwymiarowej

int [,,] LiczbyTr; // deklaracja tablicy trójwymiarowej

Tworzenie instancji tablic

- Zadeklarowanie zmiennej tablicowej nie tworzy instancji tablicy (ponieważ jest to typ referencyjny).

- Aby utworzyć instancję tablicy należy posłużyć się operatorem *new.*

- Tworząc instancję tablicy należy podać jej rozmiar w każdym z wymiarów:

```
int[] Liczby = new int[20]; // jednowymiarowa tablica 20-stoelementowa
int[,] Liczby2D = new int[2, 2]; // dwuwymiarowa tablica 2x2
```

Tworzenie instancji tablic cd.

- Jeżeli nie podamy któregokolwiek z rozmiarów w danym wymiarze, kompilator zgłosi błąd:

```
float[]   Liczby = new float[];   // BŁĄD
double[,]   LiczbyD = new double[10,   ];   // BŁĄD
```

Inicjowanie elementów tablicy

- Po utworzeniu instancji tablicy, wszystkie elementy tablicy są inicjowane domyślnymi wartościami:
  - dla typu **całkowitego** elementy inicjowane są wartościami **0**,
  - dla typu **zmiennoprzecinkowego** wartościami **0.0**,
  - dla typu **logicznego** elementy inicjowane są wartością ***false***,
  - dla typu **referencyjnego** elementy inicjowane są wartościami ***null.***

Dostęp do elementów tablicy

- Aby uzyskać dostęp do określonego elementu tablicy, należy użyć operatora indeksowania (nawiasy kwadratowe) podając indeks w danym wymiarze.

- Indeks określa położenie elementu w ciągu i może przyjmować wartości **od 0 do liczby elementów** w danym wymiarze pomniejszonym o jeden (ze względu na numerację od zera).

Dostęp do elementów - przykłady

- W przypadku tablicy jednowymiarowej w nawiasie podaje się jeden indeks:

```
int[] Liczby = new int[10];
Liczby[0] = 10; // przypisanie wartości pierwszemu elementowi
Liczby[9] = Liczby[0] + 10; // przypisanie wartości ostatniemu
```

- W przypadku tablic wielowymiarowych, indeksy dotyczące poszczególnych wymiarów, oddziela się przecinkami:

```
int[,] LiczbyD = new int[10, 10];
LiczbyD[1,2] = 100;
LiczbyD[2,1] = LiczbyD[1,2] * 2;
```

Dostęp do elementów – przykłady cd.

- W przypadku, gdy wartość podanego indeksu jest z poza zakresu (0..rozmiar-l), następuje wyrzucenie wyjątku *IndexOutOfRangeException.*

```
int[]  Liczby  =  new  int[10];
Liczby[99]  = 100;  // BŁĄD:indeks poza zakresem
```

Inicjacja elementów tablic

- W momencie tworzenia instancji tablicy, istnieje możliwość zainicjowania elementów tej tablicy wartościami początkowymi przy pomocy **listy inicjacyjnej**.

- Wartości początkowe podaje się w nawiasach {} oddzielając je przecinkami.

- Należy przy tym pamiętać o tym, aby podać wszystkie wartości początkowe.

Inicjacja elementów tablic
Lista Inicjacyjna

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

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

int[]   Liczby = new int[4]{1, 2};     // BŁĄD:
   muszą być wszystkie wartości

int[,]   LiczbyD = new int[2,  2]{{1,   3}};   //
   BŁĄD:  muszą być wszystkie wartości
```

Inicjacja elementów tablic – notacja skrócona

- Jeżeli instancja tablicy, tworzona jest w czasie deklaracji zmiennej tablicowej do inicjacji wartościami początkowymi, można użyć notacji skróconej w nawiasach {}.

- Liczba wartości podanych w tych nawiasach stanie się wtedy rozmiarem tablicy:

```
int[] Liczby = {1, 2, 3, 4}; // instancja tablicy czteroelementowej
int[] Liczby = {1, 2};  // instancja tablicy dwuelementowej
int[,] Liczby = {{1, 2}, {3, 4}};  // instancja tablicy dwuwymiarowej 2x2
```

Notacji tej można używać jedynie w czasie deklaracji zmiennej. Jeżeli zmienna jest już zadeklarowana w czasie tworzenia nowego odnośnika do instancji, notacja skrócona nie jest dozwolona.

Właściwości tablic

- Klasa *SystetmArray*, zawiera wiele przydatnych właściwości, z których można korzystać w czasie operowania na tablicach.

- Do najważniejszych i najczęściej używanych należą:

  – ***Length*** - właściwość tylko do odczytu, określająca liczbę wszystkich elementów tablicy we wszystkich wymiarach.

```
int[] Liczby = new int[10];
for (int i = 0; i < Liczby.Length; i++)
    Liczby[i] = 9999;
```

  – ***Rank*** - właściwość tylko do odczytu, określająca liczbę wymiarów tablicy.

```
int[,,] LiczbyT = new int[10, 10, 10];
if (LiczbyT.Rank == 3)
    Console.WriteLine("Tablica trójwymiarowa.");
```

Metody operujące na tablicach

- Do najważniejszych i najczęściej używanych metod klasy *SystemArray* należą:
  - ***Clear*** - (metoda statyczna) przypisuje elementom tablicy z określonego przedziału wartości domyślne (**0** dla typów całkowitych **0.0** dla typów zmiennoprzecinkowych, ***false*** dla typu ***bool*** oraz ***null*** dla typów referencyjnych).

```
int[] Liczby = {1, 2, 3, 4, 5, 6, 7, 8, 9};
System.Array.Clear(Liczby, 0, Liczby.Length); //wypełnienie
    zerami wprzedziale od 0 do Liczby.Length
```

  - ***GetLength*** - zwraca długość tablicy w określonym wymiarze.

```
int[,] LiczbyD = {{1, 2, 3, 4}, {5, 6, 7, 8}};
int rozmiar0 = LiczbyD.GetLength(0); // 2
int rozmiar1 = LiczbyD.GetLength(1); // 4
```

Metody operujące na tablicach cd.1

- ***Sort*** - (metoda statyczna) sortuje elementy tablicy. Metodę można użyć do sortowania tablicy złożonej z elementów określonej klasy czy struktury, pod warunkiem, że implementuje ona interfejs *IComparable.*

```
int[] Liczby = {4, 2, 7, 1, 5, 6, 3, 9, 8};
System.Array.Sort(Liczby); // 1, 2, 3, 4, 5, 6, 7, 8, 9
```

- ***Clone*** - tworzy nową instancję tablicy i kopiuje wartości wszystkich elementów z klonowanej tablicy. Metoda ta tworzy jedynie „płytką kopię" elementów, co oznacza, że jeżeli elementami tablicy są referencje do jakiś obiektów, kopia tej tablicy będzie zawierać referencje do tych samych obiektów co klonowana tablica.

```
int[] Liczby = {4, 2, 7, 1, 5, 6, 3, 9, 8};
int[] KopiaLiczby = (int[])Liczby.Clone(); // kopia tablicy
```

Metody operujące na tablicach cd.2

- ***IndexOf*** - zwraca indeks pierwszego elementu, którego wartość jest zgodna z wartością argumentu przekazanego do tej metody. W przypadku nie znalezienia elementu o określonej wartości w tablicy zwracana jest wartość -1:

```
int[] Liczby = {4, 2, 7, 4, 2, 7, 4, 2, 7};
int gdzie = System.Array.IndexOf(Liczby, 7); // 2
if (gdzie == -1) Console.WriteLine("Brak elementu w tablicy.");
```

- ***LastIndexOf*** - zwraca indeks ostatniego elementu, którego wartość jest zgodna z wartością argumentu przekazanego do tej metody. W przypadku nie znalezienia elementu o określonej wartości w tablicy zwracana jest wartość -1:

```
int[] Liczby = {4, 2, 7, 4, 2, 7, 4, 2, 7};
int gdzie = System.Array.LastIndexOf(Liczby, 7); // 8
if (gdzie == -1) Console.WriteLine("Brak elementu w tablicy.");
```

- ***Reverse*** - odwraca porządek elementów w tablicy.

```
int[] Liczby = {1, 2, 3, 4, 5, 6, 7, 8, 9};
System.Array.Reverse(Liczby); // 9, 8, 7, 6, 5, 4, 3, 2, 1
```

Zwracanie tablic z metod

- Tablice jak inne zmienne mogą być zwracane z metod. W takim przypadku, w sygnaturze metody, określa się typ referencyjny dla odpowiedniej tablicy, a z metody zwraca instancję tablicy.

```
class   Test
{
      static int[] UtworzTablice(int Rozmiar)
      {
          int [] tab = new int[Rozmiar]
          return tab;
      }

      static void Main()
      {
          int[] Liczby = UtworzTablice(10);
      }
}
```

Określenie rozmiaru tablicy w sygnaturze jest błędem (określamy tylko typ zwracanej wartości)

Przekazywanie tablic do metod

- Tablice mogą być przekazywane do metod w postaci argumentów. Przekazując tablicę jako argument, tworzona jest kopia zmiennej referencyjnej będąca odnośnikiem do instancji tej samej tablicy (nie jest tworzona nowa instancja tablicy, przekazanie argumentu jest natychmiastowe).

```
class Test
{
    static void Drukuj (int[] Argument)
     {
          for(int i = 0; i < Argument.Length; i++)
             Console.WriteLine(Argument[i]);
     }
     static void Main()
     {
          int[] Tablica = {0, 1, 2, 3};
          Drukuj(Tablica);
     }
}
```

Tablica argumentów Main

- W czasie uruchamiania programu pierwszą metodą jaka zostanie wykonana, jest metoda *Main*.

- Uruchamiając program z linii poleceń, można przekazać do programu dodatkowe parametry wejściowe.

- Parametry te przekazywane są do metody *Main* w postaci argumentu będącego tablicą napisów o długości równej liczbie przekazanych do programu parametrów.

Tablica argumentów Main - przykład

```
using System;
class Test
{
    static void Main(string[] args)
    {
        if (args.Length > 0)
        {
          Console.WriteLine("Parametry: ") ;
          foreach (string str in args)
             Console.WriteLine(str);
        }
        else
          Console.WriteLine("Brak parametrów.");
    }
}
```

Uruchomienie:

test.exe -x ala.txt /s:s "Ala ma kota"

Łańcuchy znaków - Klasa String

- W języku C# typ *string* jest używany do przechowywania oraz operowania na ciągu znaków w standardzie Unicode.

- Typ *string* jest aliasem do klasy *System.String*, więc obie deklaracje zmiennych mogą być stosowane zamiennie.

```
string strNapis;
System.String strInnyNapis;
```

- Po zadeklarowaniu zmiennej tego typu można przypisać jej wartość (utworzyć instancję klasy *String):*

```
string  str  =  "Test";
```

Klasa String

- Do poszczególnych znaków łańcucha można dostać się za pomocą operatora indeksowania:

```
if   (str[0]   ==   'T') { ... }
```

- Typ *string* jest jednak typem dość specyficznym. Przechowywany w łańcuchu tekst po utworzeniu instancji klasy String nie może ulec zmianie:

```
string str = "Test";
str[0]   =   'X';   // BŁĄD
```

Klasa String tworzenie napisów

- Wszelkie zmiany w łańcuchu wymagają zastosowania odpowiednich metod, właściwości i operatorów, które w rezultacie utworzą nową i zmodyfikowaną instancję klasy *String.*

```
string Napis1 =  "Jesteś", Napis2 = "Fajny", Napis3;

// utworzenie instancji Napis3 i zainicjowanie jej wartością Napis1
Napis3 = Napis1;

// utworzenie nowej instancji Napis3 i zainicjowanie jej połączoną wartością
// poprzedniej wartości Napis3, znakiem spacji i wartością Napis2
Napis3 += " " + Napis2;
```

Pola, właściwości i metody klasy String

- Klasa *String* oferuje wiele różnych użytecznych metod, pól i właściwości, z których można korzystać w czasie operowania na łańcuchach. Do najważniejszych i najczęściej używanych należą:
  - ***Empty*** - (statyczne pole) reprezentacja pustego łańcucha znaków.

```
static string Etykieta(int Numer)
{
    if (Numer == 0) return "Sól";
    if (Numer == 1) return "Cukier";
    return String.Empty;
}
```

Pola, właściwości i metody klasy String cd.1

- ***Length*** - (właściwość) zawiera aktualną liczbę znaków w łańcuchu (długość łańcucha).

```
string  str  =  "Test";
if  (str.Length  >  0)
   Console.WriteLine("Łańcuch: " + str) ;
else
   Console.WriteLine("Łańcuch jest pusty.");
```

Metody klasy String: Compare

- ***Compare*** - (metoda statyczna) porównuje leksykalnie dwa łańcuchy znaków. Porównanie odbywa się w oparciu o porządek alfabetyczny. Zwracana wartość z porównania może być:
  - ujemna (w przypadku, gdy pierwszy napis znajduje się przed drugim),
  - zerowa (gdy oba napisy są tożsame),
  - dodatnia (w przypadku, gdy pierwszy napis znajduje się za drugim).

```
string  strl   =   "Ala";
string  str2   =   "Kot";
int wyn = String.Compare(strl, str2); // ujemna wartość
```

Metoda domyślnie porównuje łańcuchy uwzględniając wielkość znaków. W przypadku, gdy wielkość znaków ma być pomijana, przy porównaniu należy użyć innej postaci metody.

Metody klasy String: Concat, Contains

- ***Concat*** - (metoda statyczna) łączy dwa lub więcej łańcuchów w jeden.

```
string  str1   =   "Ala";
string str2 = " ma kota";
string strNapis = String.Concat(str1, str2); // "Ala ma kota"

string[] napisy = {"Ala"," ma ","kota"}; //tablica łańcuchów
string Napis = String.Concat(napisy); // "Ala ma kota„
```

- ***Contains*** - (metoda) sprawdza, czy łańcuch zawiera określony napis.

```
string strl = "Te meble nie wyglądają zbyt imponująco";
string  str2   =   "meble";
bool bZawiera = strl.Contains(str2); // true
```

Metody klasy String: Copy, Format

- ***Copy*** - (metoda statyczna) tworzy nową instancję klasy *String* z identyczną zawartością.

```
string  strl   =   "Test";
string str2 = String.Copy(strl); // "Test„
```

Metody klasy String: IndexOf, IndexOfAny

- *IndexOf-* (metoda) zwraca położenie pierwszego znaku lub początku podłańcucha pasującego do wzorca w danym łańcuchu.

```
string str1 = "Te meble nie wyglądają zbyt imponująco";
string str2 = "meble";
int pos;
pos = str1.IndexOf(str2); // pos = 3
pos = strl.IndexOf('e'); // pos = 1
```

- *IndexOfAny* - (metoda) zwraca położenie któregokolwiek ze znaków spośród tych wyszukiwanych w łańcuchu.

```
string str = "Te meble nie wyglądają zbyt imponująco";
int pos = str.IndexOfAny(new char[]{'a','e','i'}); // pos = 1
```

Metody klasy String: IsNullOrEmpty, LastIndexOf

- ***LastIndexOf*** - (metoda) zwraca położenie ostatniego znaku lub początku ostatniego podłańcucha pasującego do wzorca w danym łańcuchu.

```
string strl = "Te meble nie wyglądają zbyt imponująco";
string str2 = "meble";
int pos;
pos = strl.LastIndexOf (str2); // pos = 3
pos = strl.LastIndexOf('e'); // pos = 11
```

- ***IsNullOrEmpty*** - (metoda statyczna) sprawdza, czy łańcuch nie został utworzony (czy nie utworzono instancji) lub czy istniejący łańcuch jest pusty.

```
string str1 = null;
string str2 = String.Empty;
if (String.IsNullOrEmpty(strl) && String.IsNullOrEmpty(str2) ) {
    str1 = "Test";
    str2 = "Test";
}
```

Metody klasy String: PadLeft, PadRight

- ***PadLeft*** - (metoda) uzupełnia łańcuch do zadanej długości z lewej strony zadanymi znakami (domyślnie znakami spacji). Ilość dopisanych znaków jest równa podanej wartości pomniejszonej o długość łańcucha (jeżeli jest mniejsza nic nie jest dopisywane).

```
string  str1   =   "napis1";
string  str2   =   "napis2";
strl = strl.PadLeft(15);              // "       napis1„
str2 = str2. PadLeft (15, ' . ' ) ; // „.....napis2„
```

- ***PadRight*** - (metoda) uzupełnia łańcuch do zadanej długości z prawej strony zadanymi znakami (domyślnie znakami spacji). Ilość dopisanych znaków jest równa podanej wartości pomniejszonej o długość łańcucha (jeżeli jest mniejsza nic nie jest dopisywane).

Metody klasy String: Remove, Replace

- ***Remove*** - (metoda) usuwa określoną liczbę znaków, począwszy od znaku o zadanym indeksie. Jeżeli nie zostanie określona liczba znaków, przyjmuje się, że usunięte zostaną znaki od podanego indeksu do końca łańcucha.

```
string str = "0123... 789", wynik;
wynik = str.Remove(4); // "0123"
wynik = str.Remove(4,3); // "0123789„
```

- ***Replace*** - (metoda) zamienia wszystkie wystąpienia określonego znaku lub podłańcucha znaków na zadane.

```
string str = "0123... 789", wynik;
wynik = str.Replace('.','X'); // "0123XXX789"
wynik = str.Replace("0123...","Numer "); // "Numer 789"
```

Metody klasy String: Split, Substring

- ***Split*** - (metoda) dzieli łańcuch znaków na podłańcuchy w oparciu o podane znaki separujące.

```
string str = "uruchom /a /b:cc plik.txt";
string [] Komendy = str.Split(‘ ‘) ;
foreach (string komenda in Komendy)
    Console.WriteLine(komenda);
```

Wynik:

```
uruchom
/a
/b:cc
plik.txt
```

- ***Substring*** - (metoda) zwraca fragment łańcucha znaków począwszy od elementu o określonym indeksie początkowym i określonej długości znaków (jeżeli nie zostanie określona długość, zwracany jest fragment od zadanego indeksu do końca łańcucha).

```
string str = "Te meble nie wyglądają zbyt imponująco";
string wyniki = str.Substring(3, 5); // "meble"
string wynik2 = str.Substring(28); // "imponująco"
```

Metody klasy String: ToCharArray, ToLower

- ***ToCharArray*** - (metoda) kopiuje elementy łańcucha do tablicy znaków.

```
string str = "Te meble nie wyglądają zbyt imponująco";
char[] znaki = str.ToCharArray();
```

- ***ToLower*** - (metoda) zamienia wszystkie duże litery w łańcuchu na małe.

```
string   str  =   "ALA MA KOTA";
string wynik = str.ToLower(); // "ala ma kota„
```

- ***ToUpper*** - (metoda) zamienia wszystkie małe litery w łańcuchu na duże.

```
string  str  =   "ala ma   kota";
string wynik = str.ToUpper () ; // "ALA MA KOTA"
```

Metody klasy String: Trim

- ***Trim*** - (metoda) usuwa z początku i końca łańcucha zadane znaki (domyślnie, jeżeli nie określi się jakie znaki mają być usunięte, usuwane są tylko białe znaki).

```
string  str  =   "          ...kot...     „
string wyniki = str.Trim(); // "...kot..."
string wynik2 = str.Trim(new char[]{' ','.'}); // "kot„
```

- ***TrimStart*** - (metoda) usuwa z początku łańcucha zadane znaki.

```
string  str  =   "          ...kot...     „
string wyniki = str.Trim(); // "...kot...    „
string wynik2 = str.Trim(new char[ ] {' ','.'}); // "kot...    ”
```

- ***TrimEnd*** - (metoda) usuwa z początku i końca łańcucha zadane znaki

```
string  str  =   "          ...kot.
string wyniki = str.Trim(); // "     ...kot..."
string wynik2 = str.Trim(new char[]{' ','.'}); // "     ...kot"
```

Budowanie łańcuchów - klasa StringBuilder

- Klasa *StringBuilder* została stworzona w celu modyfikowania łańcuchów znaków, bez potrzeby tworzenia nowej instancji klasy.

- Wszelkie zmiany wykonywane na instancji klasy *StringBuilder* modyfikują łańcuch bezpośrednio.

- Aby móc operować na łańcuchu za pomocą metod i właściwości tej klasy, należy utworzyć jej instancję.

```
string napis = "Testowy napis";
StringBuilder sbn = new StringBuilder(napis);
```

Właściwości *StringBuilder: Capacity, Length*

- ***Capacity*** - (właściwość) pozwala odczytać lub ustawić maksymalną liczbę znaków, jaka może być przechowywana w pamięci zaalokowanej dla obiektu.

```
string napis = "Testowy napis";
StringBuilder sbn = new StringBuilder(napis);
sbn.Capacity = 50; // maksymalnie 50 znaków
```

- ***Length*** - (właściwość) pozwala odczytać lub ustawić aktualną liczbę znaków przechowywanych w pamięci.

```
string napis = "Testowy napis";
StringBuilder sbn = new StringBuilder(napis);
sbn.Length = 5;
napis = sbn.ToString(); // "Testo"
```

Metody StringBuilder: Append, AppendFormat

- ***Append*** - (metoda) pozwala na dołączenie do łańcucha napisowej reprezentacji dowolnego typu danych.

```
string napis = "Testowy napis: ";
StringBuilder sbn = new StringBuilder(napis);
sbn.Append(1700); // "Testowy napis: 1700”
sbn.Append(8.8d); // "Testowy napis: 17008,8”
```

Metody StringBuilder: AppendLine, Insert

- ***AppendLine*** - (metoda) pozwala dołączyć do łańcucha znak przejścia do nowej linii.

```
string napis = "Cena";
StringBuilder sbn = new StringBuilder(napis);
sbn.AppendLine();
sbn.Append("Inna cena");
Console.WriteLine(sbn.ToString());
```

Wynik:
Cena
Inna cena

- ***Insert*** - (metoda) pozwala wstawić w odpowiednim miejscu do łańcucha napisową reprezentację dowolnego typu danych.

```
string napis = "Testowy napis: ";
StringBuilder sbn = new StringBuilder(napis);
sbn.Insert(2, 1700); // "Te1700stowy napis: "
sbn.Insert(7, 8.8d); // "Te1700s8,8towy napis: "
```

Metody StringBuilder: Remove, Replace

- ***Remove*** - (metoda) pozwala usunąć z łańcucha określoną liczbę znaków, począwszy od zadanego indeksu początkowego.

```
string napis = "Testowy napis: ";
StringBuilder sbn = new StringBuilder(napis);
sbn.Remove(7, 8); // "Testowy„
```

- ***Replace*** - (metoda) pozwala na zamianę wszystkich wystąpień znaków lub podłańcuchów wewnątrz łańcucha.

```
string napis = "Testowy napis: ";
StringBuilder sbn = new StringBuilder(napis);
sbn.Replace(":", " ->"); // "Testowy napis -> "
```

Kolekcje

- W języku C# kolekcja jest grupą elementów, w której każdy element jest obiektem.

- W .NET Framework można znaleźć szeroką gamę różnego rodzaju kolekcji: list, kolejek, stosów czy słowników.

- Kolekcje znajdują się w przestrzeni nazw *System.Collections.*

Kolekcje - klasy

- Do najpopularniejszych klas definiujących kolekcje z przestrzeni *System.Collections* należą:
  - *ArrayList* - reprezentuje tablicę, której rozmiar może zostać zwiększony, kiedy zajdzie taka potrzeba;
  - *Hashtable* - reprezentuje kolekcję par klucz/wartość;
  - *Queue* - reprezentuje kolejkę (FIFO - first in, first out);
  - *SortedList* - reprezentuje listę posortowanych elementów;
  - *Stack* - reprezentuje stos (LIFO - last in, first out).

Klasa ArrayList

- ArrayList można traktować jak ulepszenie tablicy.
- Elementy przechowywane w obiekcie klasy *ArrayList* mogą być dowolnego typu.

Właściwości ArrayList:

- ***Count*** - (właściwość) pozwala odczytać aktualną liczbę elementów kolekcji.

```
ArrayList arl = new ArrayList(10);
if (arl.Count > 0)
{
    ....
}
```

Metody ArrayList: Add, AddRange

- ***Add*** - (metoda) dodaje element na końcu kolekcji.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("ala") ;
wyrazy.Add("ma");
wyrazy.Add("kota") ;
```

- ***AddRange*** - (metoda) pozwala dołączyć do kolekcji inną kolekcję elementów.

```
ArrayList wyrazy1 = new ArrayList();
ArrayList wyrazy2 = new ArrayList();
wyrazy1.Add("ala") ;
wyrazy2.Add("ma") ;
wyrazy2.Add("kota") ;
wyrazy1.AddRange(wyrazy2); // dodanie zawartości kolekcji wyrazy2
```

Metody ArrayList: Clear, Contains

- ***Clear*** - (metoda) usuwa wszystkie elementy z kolekcji.

```
wyrazy.Clear() ; // usunięcie elementów
```

- ***Contains*** - (metoda) pozwala określić, czy w kolekcji znajduje się element.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("ala");
wyrazy.Add("ma");
wyrazy.Add("kota") ;
if (wyrazy.Contains( "ma" ) ) // sprawdzenie czy zawiera wyraz
{
    .....
}
```

Metody ArrayList: GetRange, IndexOf

- ***GetRange*** - (metoda) zwraca podzbiór elementów danej kolekcji. Zwrócony podzbiór zawiera określoną liczbę elementów, począwszy od określonego indeksu (indeks podaje się jako pierwszy argument, a długość jako drugi).

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("ala") ;
wyrazy.Add("ma");
wyrazy.Add("kota") ;
ArrayList podzbiór = wyrazy.GetRange(1, 2); // "ma", "kota„
```

- ***IndexOf*** - (metoda) zwraca numer indeksu pierwszego elementu pasującego do wyszukiwanego wzorca.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("kot"); wyrazy.Add("pies") ; wyrazy.Add("pies") ;
int pos = wyrazy.IndexOf("pies") ; // pos = 1
```

Metody ArrayList: Insert, InsertRange

- ***Insert*** - (metoda) pozwala na wstawienie do kolekcji elementu na określoną indeksem pozycję.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("kot");
wyrazy.Add("pies") ;
wyrazy.Insert(0, "mysz"); // wyrazy = {"mysz", "kot", "pies")
```

- ***InsertRange*** - (metoda) pozwala na wstawienie do kolekcji elementów na określoną indeksem pozycję z innej kolekcji.

```
ArrayList wyrazy1 = new ArrayList();
wyrazy1.Add("kot") ;
wyrazy1.Add("pies") ;
ArrayList wyrazy2 = new ArrayList();
wyrazy2.Add("mysz") ;
wyrazy1.InsertRange(0, wyrazy2); // wyrazy1 = {"mysz", "kot", "pies")
```

Metody ArrayList: Remove, RemoveAt, RemoreRange

- ***Remove*** - (metoda) usuwa z kolekcji pierwszy element pasujący do wzorca.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("kot") ;
wyrazy.Add("pies") ;
wyrazy.Add("krowa") ;
wyrazy.Add("pies") ;
wyrazy.Remove("pies"); // wyrazy = {"kot", "krowa", "pies")
```

- ***RemoveAt*** - (metoda) usuwa z kolekcji element o określonym indeksie.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("kot") ;
wyrazy.Add("pies") ;
wyrazy.Add("krowa") ;
wyrazy.Add("pies") ;
wyrazy.RemoveAt(1); // wyrazy = {"kot", "krowa", "pies")
```

- ***RemoveRange*** - (metoda) usuwa z kolekcji określoną ilość elementów, począwszy od określonego indeksu początkowego.

Metody ArrayList - pozostałe metody

- ***Reverse*** - (metoda) odwraca porządek (kolejność) elementów w kolekcji.

- ***SetRange*** - (metoda) kopiuje do kolekcji elementy innej kolekcji nadpisując istniejące, począwszy od określonego indeksu.

- ***Sort*** - (metoda) sortuje elementy w kolekcji.

```
ArrayList wyrazy = new ArrayList();
wyrazy.Add("mysz") ;
wyrazy.Add("pies") ;
wyrazy.Add("kot") ;
wyrazy.Sort(); // wyrazy = {"kot", "mysz", "pies")
```

- ***ToArray*** - (metoda) kopiuje elementy kolekcji do nowej tablicy.

Klasa Hashtable

- Klasa ***Hashtable*** oferuje wiele różnych metod i właściwości, z których można korzystać w czasie operowania na tablicach haszujących.

- Przy dodawaniu elementu do tablicy z haszowaniem należy podać nie tylko dany element, ale także unikalny klucz, poprzez który będziemy uzyskiwali dostęp do tego elementu.

- Zarówno klucz jak i element mogą być obiektami dowolnego typu.

- Do tablicy z haszowaniem elementy dodawane są za pośrednictwem metody **Add()**.

Klasa Hashtable przykład użycia

```
using System;
using System.Collections;
public class HashtableDemo
{
  private static Hashtable ages = new Hashtable();
  public static void Main()
  {
      // Dodanie elementów do tablicy z haszowaniem,
      // każdemu elementowi przypisywany jest łańcuch znaków
      ages.Add("Scott", 25);
      ages.Add("Sam", 6);
      ages.Add("Jisun", 25);

      // Uzyskanie dostępu do elementu o podanym kluczu
      if (ages.ContainsKey("Scott"))
      {
          int scottsAge = (int) ages["Scott"];
          Console.WriteLine("Scott ma tyle lat: " + scottsAge.ToString());
      }
      else
          Console.WriteLine("Danych Scotta nie ma w tablicy...");
  }
}
```

Klasa Queue

- Klasa *Queue* oferuje wiele różnych metod i właściwości, z których można korzystać w czasie operowania na kolejkach FIFO.

Klasa Queue – właściwości, metody

- *Count* - (właściwość) pozwala odczytać aktualną liczbę elementów kolekcji.
- *Clear* - (metoda) usuwa zawartość kolekcji.
- *Contains* - (metoda) sprawdza czy w kolekcji znajduje się określony element.
- *Dequeue* - (metoda) usuwa element z kolekcji i zwraca jego wartość (jeżeli nie ma żadnego elementu w kolekcji, wywołanie tej metody zwróci wyjątek).
- *Enqueue* - (metoda) dodaje element do kolejki.
- *Peek* - (metoda) zwraca wartość pierwszego elementu w kolejce nie usuwając go.
- *ToArray* - (metoda) kopiuje elementy kolekcji do nowej tablicy.

Klasa Queue - przykład

```
using System;
using System.Collections.Generic;

class Example
{
   public static void Main()
   {
      Queue<string> numbers = new Queue<string>();
      numbers.Enqueue("one");
      numbers.Enqueue("two");
      numbers.Enqueue("three");
      numbers.Enqueue("four");
      numbers.Enqueue("five");

      foreach( string number in numbers )
      {
         Console.WriteLine(number);
      }
     string element = numbers.Dequeue();
     Console.WriteLine(element );
   }
}
```

Klasa SortedList

- Klasa ta reprezentuje kolekcję (klucz i wartości) posortowanych zgodnie z kluczami i dostępnych poprzez klucz i indeks.

- Klasa ta jest hybrydą tablic haszujących (Hashtable) i tablic (ArrayList).

Klasa SortedList – właściwości, metody

- ***Count*** - (właściwość) pozwala odczytać aktualną liczbę elementów kolekcji.
- ***Keys*** - (właściwość) zawiera kolekcję kluczy listy uporządkowanej.
- ***Values*** - (właściwość) zawiera kolekcję wartości listy uporządkowanej.
- ***Add*** - (metoda) dodaje element ze specyficznym kluczem i wartością na końcu kolekcji.
- ***Clear*** - (metoda) usuwa zawartość kolekcji.
- ***ContainsKey*** - (metoda) sprawdza czy kolekcja zawiera określony klucz.
- ***ContainsValue*** - (metoda) sprawdza czy kolekcja zawiera określoną wartość.
- ***GetByIndex*** - (metoda) zwraca wartość elementu kolekcji o określonym indeksie.
- ***GetKey*** - (metoda) zwraca klucz elementu kolekcji o określonym indeksie.

Klasa SortedList – przykład użycia

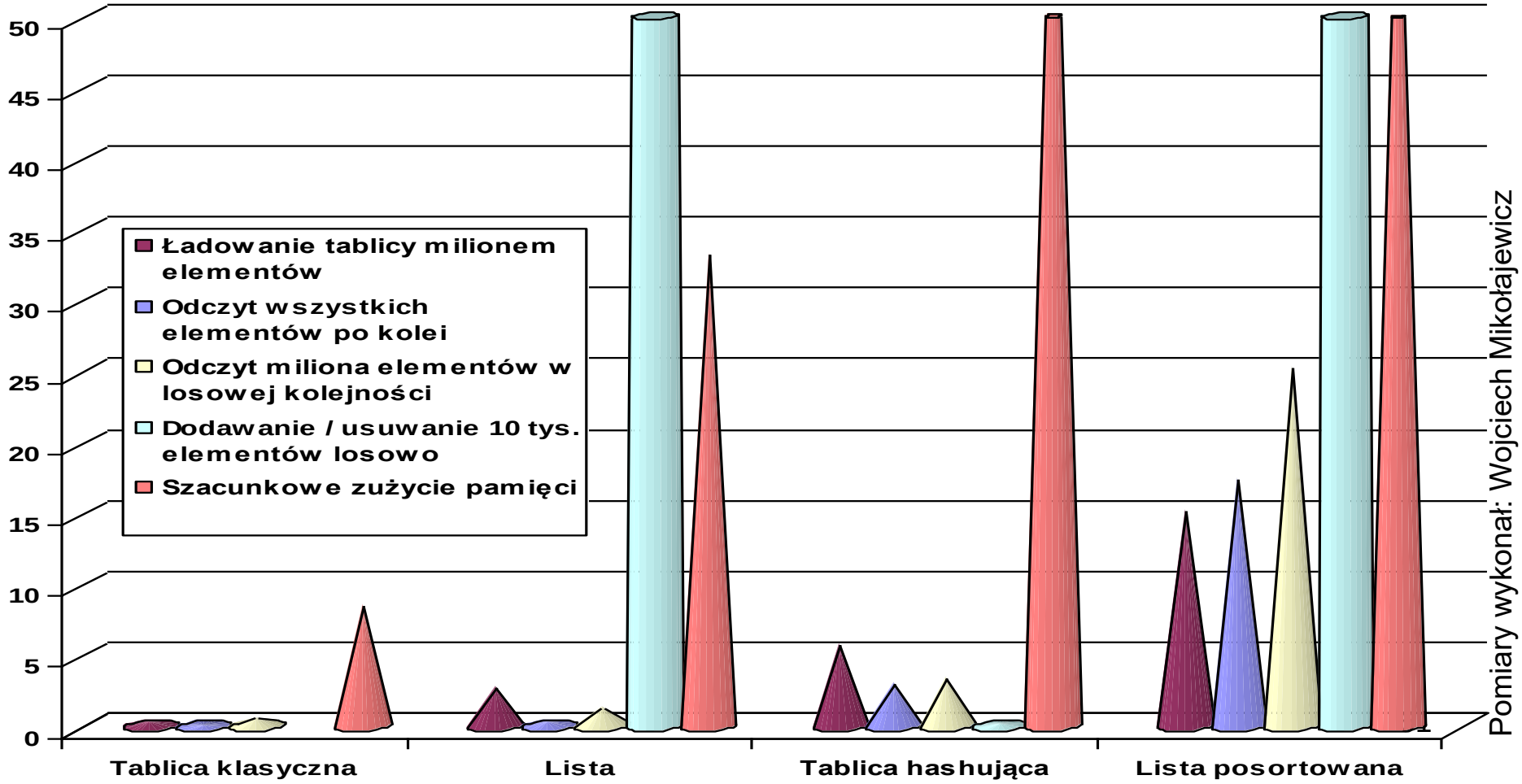
```
using System;
using System.Collections.Generic;

class Example
{
   public static void Main()
   {
         SortedList sit = new SortedList();
         sit.Add("Ala", "kot");
         sit.Add(„Zbyszek", "pies");
         sit.Add("Ania", „krowa");

         foreach (object obj in sit.Keys)
            Console.WriteLine(obj.ToString());
     }
}
```

Porównanie czasu wykonania na różnych typach tablic

	[s]	[s]	[s]	[s]	[MB]
Tablica klasyczna	0,261	0,030	0,563		8,5
Lista	2,710	0,201	1,335	343,872	33,3
Tablica hashująca	5,712	2,946	3,357	0,080	70,0
Lista posortowana	15,180	17,444	25,228	381,501	64,2



To już jest koniec ....