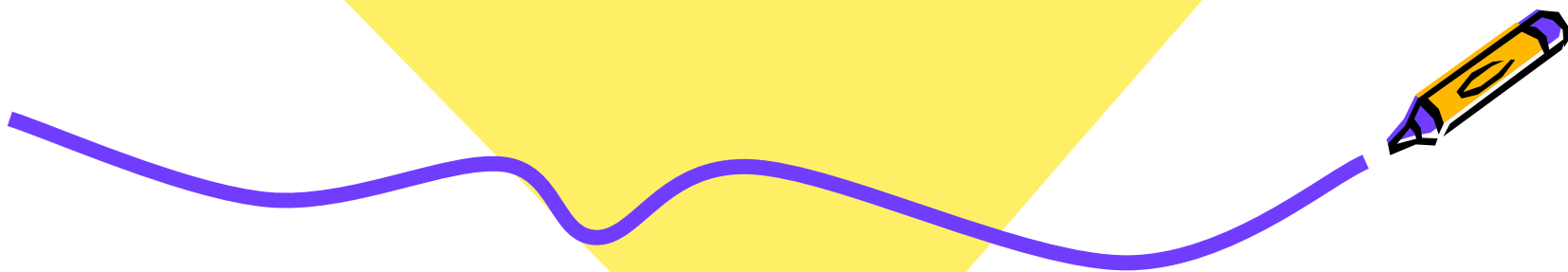




Лекция 7. Массиви, Колекции.



Какво е колекции?

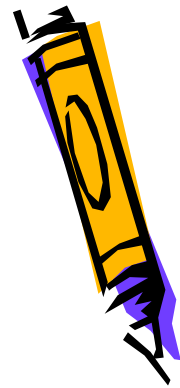
- Колекция наричаме група от взаимосвързани елементи.
- Като тип, колекцията предоставя възможност за съхраняване и достъп до елементите в нея.



Колекция ?

Според Наков:

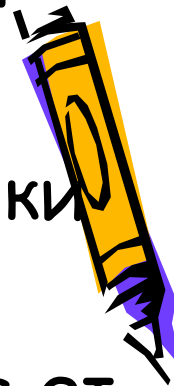
- Колекциите са известни още като "класове - контейнери", "контейнер класове" или само "контейнери".
- Те са абстрактни типове данни (ADT) и могат да бъдат имплементирани по различен начин, например: чрез масив, чрез свързан списък, чрез различни видове дървета, чрез пирамида, чрез хеш-таблица и т. н.
- Колекциите могат да бъдат с фиксиран размер (такива са например масивите) или с променлив размер (такива са например свързаните списъци, тоест arraylist).
- Колекциите могат да бъдат само за четене или да позволяват и промени.



Класовете, имплементиращи колекции в FCL са:

- Клас **System.Array**, явяващ се базов за всички видове масиви.
- Класове, които се намират в пространствата от имена **System.Collections** (**Hashtable**, **ArrayList** и др.) или **System.Collection.Specialized**.

Забележка: Независимо, че клас **Array** не е част от именовано пространство **System.Collections**, той е колекция, тъй като имплементира **ICollection** интерфейс.



Масивите и останалите колекции:

- Масивът е **хомогенна, силно типизирана** колекция, с **фиксиран размер** (размерът се указва при инициализирането им и определя броя на елементите им).
- Елементите на масива са от един и същи тип (хомогенна колекция) и могат да бъдат както от примитивен стойностен тип, така и от референтен тип (инстанции на класове дефинирани от нас).
- За сравнение: Елементите на колекцията са обекти. Работата с колекция е свързана с конвертирания - неявни при добавяне на елемент и явни - при извличане на елемент.



Масиви в Net Framework.

- Базов клас за всички масиви в Net Framework е абстрактният клас **System.Array**.
- Това, че всички масиви в .NET Framework наследяват (явно или неявно) типа **System.Array**, означава, че те са референтни типове и стойностите им се разполагат в блокове от динамичната памет (т. нар. managed heap).
- В .NET Framework се поддържат едномерни, многомерни и jagged arrays, тоест масиви от масиви ("назъбени" масиви).
- Достъпът до елементите им става по индекс (пореден номер на елемента).
- Всеки от тези видове пази в себе си информация за броя на размерностите си (т. нар. **ранг**), както и границите на всяка от тях.
- CLR е оптимизиран за работа с едномерни, нулево-базирани масиви, затова се препоръчва тяхното използване, когато е възможно. Масивите могат да се инициализират при деклариране.



Какво се получава, ако имаме само декларация на масив? Нека имаме:

```
int[] n;
```

- В стека се заделя променливата `n`, която се инициализира със стойност `null`.
- Нека имаме деклариране и заделяне (създаване) на масив:

```
int[] n = new int[5];
```

- Вече в стека се заделя променливата `n`, която сочи към участък от 5 елемента в динамичната памет (heap), които се инициализират със стойност 0 (тъй като елементите на масива са от `int` тип).



```
int[] numbers = new int[] {1,4,66,4,7};
```

- В стека се заделя променливата `numbers`, която сочи към участък от 5 елемента в динамичната памет, където се съхраняват целите стойности на елементите на масива.

- Нека имаме деклариране, заделяне (създаване) и инициализиране на масив с елементи от `value` тип.

```
string[] StringsSet = new string[] {"Varna",  
    "Sofia", "Russe", "Burgas"};
```

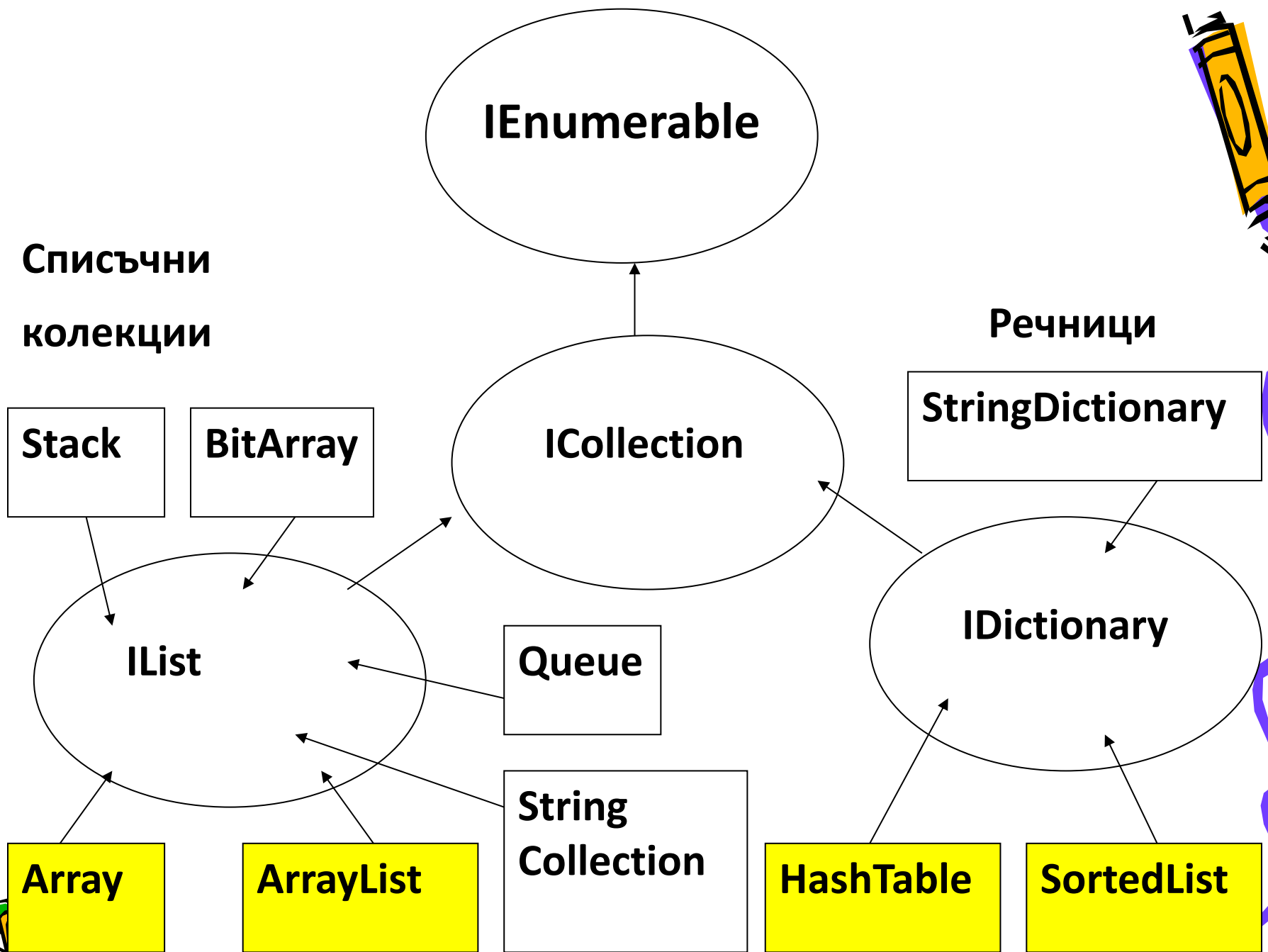
- В стека се заделя променливата `StringsSet`, която ще сочи към участък от 4 елемента в динамичната памет - елементи от тип `string`. Тъй като всеки елемент на масива от своя страна е обект от референтен тип, следователно всеки елемент от `heap` -а ще сочи от своя страна към друга област от динамичната памет, където ще се пази стойността му.



Масиви в Net Framework.

- Като референтни типове, масивите се предават по референция (т.е. по адрес, а не по стойност). Ако искаме да подадем даден масив като параметър, но да го защитим от промяна стойностите на неговите елементи, трябва да подадем негово копие. Копие на масив можем да направим чрез статичния метод **Array.Copy(...)**. Този статичен метод прави плитко копие на елементите на масива. Това означава, че ако копираме масив от референтни типове, елементите на новия масив ще сочат към същото място в паметта, към което са сочили съответно елементите от стария масив.
- От своя страна типът **System.Array** имплементира следните интерфейси: **ICloneable**, **IList**, **IEnumerable** и **ICollection**, които позволят масивите да се използват лесно в различни ситуации.





- **ICloneable** - По подразбиране масивите се копират плитко (shallow copy). Този интерфейс предоставя метод **Clone()**, който се използва за клониране на масив, тоест създава се нова инстанция на масива, със стойности, като на оригинала.
- **IList** - наследява интерфейс **ICollection** и определя колекция, към елементите на която може да се получи достъп чрез индекс, започвайки от 0. Типът **System.Array** експлицитно (явно) имплементира всеки един от методите на **IList** (и **IEnumerable**).
- **IDictionary** - Определя колекция речник, която се състои от двойки: ключ/стойност
- **IEnumerator**, който **System.Array** наследява предоставя единствено метода **GetEnumerator()**, поддържащ изброител за който и да е тип колекция. Изброителят обезпечава стандартизиран начин за поелементен достъп към съдържанието на една колекция, в това число и достъп чрез цикъл **foreach**.
- **IEnumerator** - Съдържа свойство **Current** и методи **MoveNext()** и **Reset()**, които позволяват поелементно извличане на съдържанието на колекцията



- **IDictionaryEnumerator** - Определя изброител за колекции, които реализират интерфейс **IDictionary**.
- **ICollection** - наследява **IEnumerable** и дефинира основни методи и свойства без които не може нито една колекция, например осигурява свойството **Count** (размер) и средства за синхронизация на достъпа до елементите:
 - **int Count {get;}** - Определя количеството на елементите на колекцията в дадения момент. Ако **Count** е нула, то колекцията е празна.
 - **void CopyTo (Array target, int startIdx)** - Метод, който обезпечава преход от колекция към стандартен C#-масив, копирайки съдържанието на колекцията в масив, зададен чрез параметър **target**, започвайки с индекс, зададен чрез параметър **startIdx**.
- **IComparer** - Определя метод **Compare()**, който изпълнява сравнение на обекти, съхранявани в колекцията.



В добавка към методите, определени в интерфейс `ICollection`, интерфейс `IList` определя и собствени методи:

- `int Add(object obj)`- Чрез този метод се добавя обект `obj` в колекция. Връща индекса на съхранения обект.
- `void Clear()` - Метод, чрез който се изтриват елементите на колекцията
- `bool Contains(object obj)`- метод, който връща `true`, ако в колекцията се съдържа обект `obj`, и `false` - в противен случай
- `int IndexOf(object obj)` - Връща индекса на обекта `obj`, ако той (обекта) се съдържа в колекцията. Ако `obj` не бъде намерен, то методът връща `-1`.
- `void Insert(int idx, object obj)`- Вмъква в колекцията обект `obj` по индекс, зададен чрез параметър `idx`. Елементите, намиращи се до и елемента с индекс `idx` се придвижват напред, за да освободят място на вмъкнатия обект `obj`.



- `void Remove(object obj)` - Отстранява първия срещнат елемент `obj` от извиканата колекция. Елементите, намиращи се до `obj` се преместват назад, за да ликвидират образувалата се празнина.
- `void RemoveAt(int idx)` - Отстранява се от колекцията обект, по зададен индекс - `idx`. Елементите до отстранения елемент се преместват назад, за да ликвидират образувалата се празнина.
- `bool IsFixedSize { get; }` - Свойство, което има стойност `true`, ако колекцията е с фиксиран размер. Това означава, че в такава колекция не бива да се вмъкват и изтриват елементи.
- `bool IsReadOnly { get; }` - Свойство, което приема стойност `true`, ако колекцията е само за четене.
- `object this[int idx] { get; set; }` - Индексатор, който се използва за прочитане или запис на елемент с индекс `idx`. Не бива да се използва за добавяне на нов елемент в колекция.



- Абстрактният тип **System.Array** предлага голямо количество public методи и свойства, които също се наследяват от всички масиви.
- В следващите таблици ще разгледаме по-важните свойства на **System.Array** и static методи за създаване, реверсиране, търсене и сортиране и др.



Свойства

Свойство	Описание
Rank	<p>Връща броя на размерностите (измеренията) на масива:</p> <pre>string[] StringsSet = new string[] { "Varna", "Sofia", "Russe", "Burgas" }; int l = StringsSet.Rank; Console.WriteLine("StringsSet.Rank={0}", l); //StringsSet.Rank=1</pre>
Length	<p>Връща цяло число от тип <code>System.Int32</code>, което представлява общия брой на елементите (от всички размерности) на масива.</p> <p>Пример:</p> <pre>string[] StringsSet = new string[] { "Varna", "Sofia", "Russe", "Burgas" }; int l = StringsSet.Length; Console.WriteLine("StringsSet.Length={0}", l); ; //StringsSet.Length=4</pre>
LongLength	<p>Връща цяло число от тип <code>System.Int64</code>, което представлява общия брой на елементите (от всички размерности) на масива.</p> <pre>int[,] m1 = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } }; long l = m1.LongLength; Console.WriteLine("m1.LongLength={0}", l); //m1.LongLength=8</pre>
IsFixedSize	<p>Връща <code>bool</code> стойност, която показва дали масива е с фиксиран размер. По подразбиране <code>false</code>.</p> <pre>int[,] m1 = new int[4, 2] { { 1, 2 },</pre>



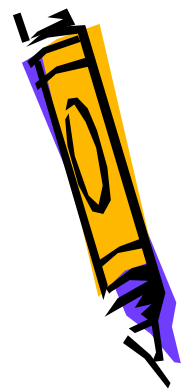
IsReadOnly

Връща bool стойност, която показва дали масива е само за четене. По подразбиране false.

```
int[,] m1 = new int[4, 2]
{
    {1,2},
    {3,4},
    {5,6},
    {7,8}
};
```

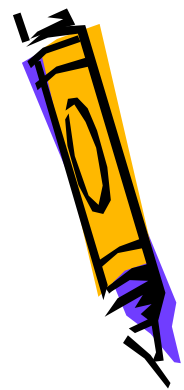
```
bool m = m1.IsReadOnly;
Console.WriteLine("m1.IsReadOnly={0}",m);
//m1.IsReadOnly=False
```

Забележка: Първите 3 свойства са достъпни само за четене.



Методи

Метод	Описание
Sort(...)	<p>Статичен метод на <code>System.Array</code>. Сортира елементите на даден масив по големина: Пример: <code>Array.Sort(StringsSet);</code></p> <p>- <code>Array.Sort(<Array_name>)</code> – сортира елементите на зададения едномерен масив, като очаква те да имплементират интерфейса <code>IComparable</code>. Този интерфейс е имплементиран (no default) от много стандартни типове – <code>Int32</code>, <code>Float</code>, <code>Double</code>, <code>Decimal</code>, <code>String</code>, <code>DateTime</code> и др.</p> <p>- Ако сме си дефинирали наш тип (клас, структура) и искаме да сортираме масив от елементи от потребителски тип, то трябва имплементираме интерфейс <code>IComparable</code> и по-точно неговия виртуален метод <code>CompareTo(...)</code>. Ако използваме <code>IComparer</code> интерфейс, трябва да имплементираме неговия виртуален метод <code>Compare()</code>. Тогава трябва да използваме следната дефиниция на <code>Sort</code>: <code>Array.Sort(<Array_name>, IComparer_Compare);</code> сортира елементите на дадения едномерен масив по зададена схема за сравнение (имплементирана в интерфейса <code>IComparer</code>).</p>
Reverse (...)	<p>Статичен метод на <code>System.Array</code>. Обръща елементите на даден едномерен масив в обратен ред. Ако масивът не е едномерен се подава <code>RankException</code>.</p> <p>Да се реверсва масива: <code>Array.Reverse(<Array_name>);</code> Пример: <code>Array.Reverse(StringsSet);</code></p>
BinarySearch (...)	<p>Статичен метод на <code>System.Array</code>. Търси за даден елемент в даден масив чрез метода на двоичното търсене. Методът предполага, че елементите на масива се подредени по големина предварително.</p> <p>- Да се изведе индекса на елемент с определена стойност, например "Russe". Масивът трябва да бъде предварително сортиран, за да са коректни резултатите от <code>BinarySearch</code>: <code>int j = Array.BinarySearch(<Array_name>, <Array_value>);</code> Пример: <code>int j = Array.BinarySearch(StringsSet, "Russe");</code></p>



GetLength(...)

Връща броя на елементите в зададено измерение на масива. (Instance метод).

```
int[,] m1 = new int[4, 2]
{
    {1,2},
    {3,4},
    {5,6},
    {7,8}
};
```

```
Console.WriteLine(m1.GetLength(0));
//m1.GetLength(0)=4
//m1.GetLength(1)=2
```

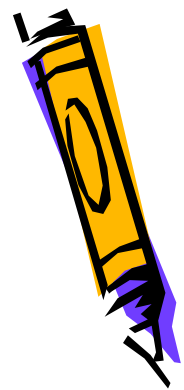
(Instance метод). Връща имплементация на интерфейса `IEnumerator` за елементите на масива. Това дава възможност да се използва конструкцията `foreach(...)`, чрез която може да се обхождат всички елементи на масива. Обхождането за многомерни масиви става отляво на дясно по размерностите, т.е. най-дясното измерение се сменя най-бързо.

Пример: Репечатване на едномерен масив `myArr`:

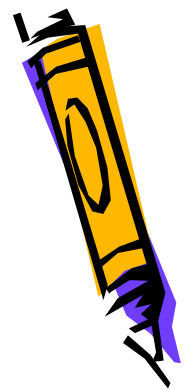
```
GetEnumerator() string[] myArr = new string[] {"Varna",
                                             "Sofia", "Russe", "Burgas"};
```

```
System.Collections.IEnumerator myEnumerator
= myArr.GetEnumerator();
```

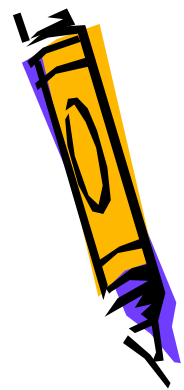
```
while (myEnumerator.MoveNext())
{
    Console.Write("\t{0}",
myEnumerator.Current);
}
```



Clear(...)	<p>Статичен метод на System.Array. Задава стойност 0 (null за референтни типове) на елементите в зададен интервал.</p> <p>Пример:</p> <pre>string[] StringsSet = new string[] {"Varna", "Sofia", "Russe", "Burgas"}; Array.Clear(StringsSet,0,2);</pre> <p>Изтрива от измерение с индекс 0 на масива StringsSet 2 елемента. Остават в случая "Russe", "Burgas".</p>
IndexOf(...)	<p>Статичен метод на System.Array. Връща индекса на първото срещане на дадена стойност в даден едномерен масив. Ако елементът не се среща в масива, връща -1. Ако масивът не е едномерен, се подава RankException.</p> <pre>string[] StringsSet = new string[] {"Varna", "Sofia", "Russe", "Burgas"}; int t = Array.IndexOf(StringsSet,"Russe"); Console.WriteLine(t);//2</pre>
LastIndexOf(...)	<p>Статичен метод на System.Array. Връща индекса на последното срещане на дадена стойност в даден едномерен масив. Ако елементът не се среща в масива връща -1. Ако масивът не е едномерен се подава RankException.</p> <pre>Console.WriteLine(Array.LastIndexOf(StringsSet, "Russe")); //2</pre>



CreateInstance(...)	<p>Статичен метод на System.Array. Създава динамично (по време на изпълнение) инстанция на типа System.Array, като може да се зададе тип на елементите (в примера по-долу int, брой размерности, в случая 1, долна граница и брой елементи за всяка размерност, в случая - 5 елемента.</p> <pre>Array my1DArray = Array.CreateInstance(typeof(Int32), 5); for (int i = my1DArray.GetLowerBound(0); i <= my1DArray.GetUpperBound(0); i++) my1DArray.SetValue(i + 1, i); //1 2 3 4 5</pre>
GetLowerBound()	(Instance метод). Виж горния пример. Връща долната граница на специфицирана дименсия.
GetUpperBound()	(Instance метод). Виж горния пример. Връща горната граница на специфицирана дименсия.
SetValue()	(Instance метод). Виж горния пример. Задава на определен елемент на масива i определена стойност - i+1.
GetValue()	<p>(Instance метод). Връща стойност на елемент на масива, при специфициране на индекса:</p> <pre>Array my1DArray = Array.CreateInstance(typeof(Int32), 5); for (int i = my1DArray.GetLowerBound(0); i <= my1DArray.GetUpperBound(0); i++) {my1DArray.SetValue(i + 1, i); Console.WriteLine(my1DArray.GetValue(i)); //1 2 3 4 5 }</pre>
Copy(...)	<p>Статичен метод на System.Array. Копира елементите на един масив (или част от тях) в друг масив. Този метод извършва, ако е необходимо, преобразуване на типовете на масивите.</p> <pre>string[] myArr = new string[] {"Varna", "Sofia", "Russe", "Burgas"}; string[] narr = new string[4]; Array.Copy(myArr, narr, 2); // narr = {"Varna", "Sofia"}</pre>



Специфика на работа с различни видове масиви C#:

- едномерен масив



Пример 1. Деклариране на масив с елементи от value тип, инстанциране (с new) и инициализация:

- //Вариант 1.

```
int[] primes;
```

```
//Деклариране на отворен масив
```

```
primes = new int [3];
```

```
//Инстанциране - с new + задаване на  
размер
```



//или Вариант 2

```
int[] primes1 = new int[3];
```

//или Вариант 3

```
const int COUNT = 3;
```

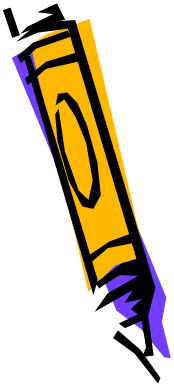
```
int[] primes1 = new int[COUNT];
```

//инициализация чрез for

```
for (int i=0;i<primes.Length;i++)
```

// Length - брой елементи

```
primes[i] = int.Parse(Console.ReadLine());
```



// или Деклариране на масив,
инстанциране и инициализация
едновременно:

```
string[] names = { "Mimi", "Lili" };
```

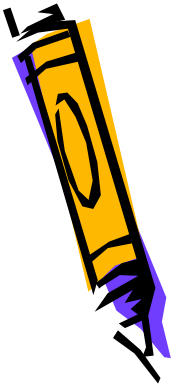
//едномерен масив

```
int[] primes = {2, 3, 5, 7, 11, 13, 17, 19};
```

//едномерен масив

```
int[] primes = new int [] {2, 3, 5, 7, 11, 13, 17,  
19};
```

```
int[] primes = new int [8]{2, 3, 5, 7, 11, 13,  
17, 19};
```



//Обхождане с foreach:

```
foreach (int p in primes)
```

```
Console.Write("{0} ", p);
```

```
// Output: 2 3 5 7 11 13 17 19
```

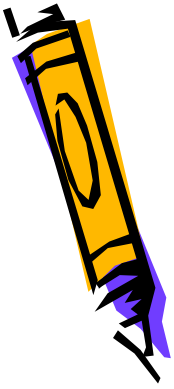
**!!! Използването на foreach е възможно,
защото масивите реализират
интерфейса IEnumerable**

//Промяна на стойността.

```
for (int i = 0; i < primes.Length; i++)
```

```
{ primes[i] = primes[i] * primes[i];}
```

```
// Output: 4 9 25 49 121 169 289 361
```

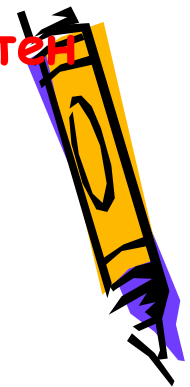


- **Пример 2. Деклариране на масив с елементи от референтен тип, а именно клас Book и обхождане с foreach:**

```
class Book
{
    public Book(string name)
    {
        this.Name = name;
    }
    public string Name { get; set; }
}
```

```
static void Main()
{
    Book[] book = new Book[3] { new Book("Червено и черно"),
    new Book("Война и мир"), new Book("Евгений Онегин") };

    foreach (Book b in book)
    {
        Console.WriteLine(b.Name);
    }
    Console.ReadLine();
}
```



Многомерни масиви

Освен вече разгледаните едномерни масиви, .NET Framework поддържа и многомерни такива (масиви с няколко размерности).

Декларирането на многомерен масив е почти същото, както при едномерните, но само с една разлика – трябва да поставим запетая между размерностите му:

```
int[,] matrix = new int[3,3];  
char[,] box = new char[2,5,10];
```



Инициализиране и достъп до елементите

Ако искаме да инициализираме многомерен масив още при декларация трябва да спазим следното правило, а именно - да поставим всяко измерение в отделни "къдрави скоби":

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

Достъпът до елементите отново е по индекс, само че в многомерния вариант отново трябва да поставим запетая между индексите на отделните размерности:

```
int elem = matrix[1,2]; //6
```



1	2	3
4	5	6
7	8	9



Пример:

```
int[,] m1 = new int[4,2]
```

```
    {{1,2},{3,4},{15,6},{7,888}};
```

```
for (int row = 0; row < m1.GetLength(0); row++)
```

```
for (int col = 0; col < m1.GetLength(1); col++)
```

```
    Console.Write("{0} ", m1[row, col]);
```

```
//1 2 3 4 15 6 7 888
```

`GetLength(0)` – извлича размерността на първото измерение

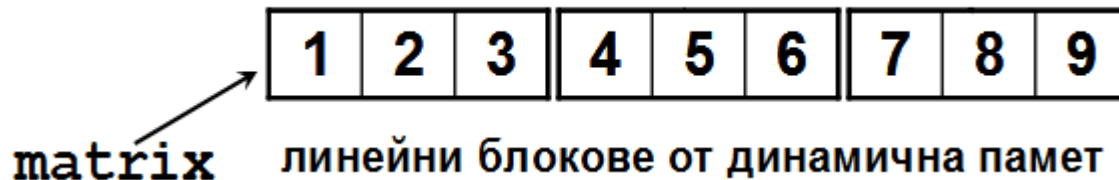


Разположение в паметта

- Многомерните масиви разполагат елементите си последователно – един след друг в линейни блокове от динамичната памет.
- Ето как би изглеждало това за вече деклариран и инициализиран с естествените числа от 1 до 9 двумерен масив **matrix**:
- `int[,] matrix =`

`{ {1, 2, 3} , {4, 5, 6} , {7, 8, 9} };`

1	2	3
4	5	6
7	8	9



Масиви от масиви (jagged масиви)

- В .NET Framework могат да се използват още масиви от масиви или т. нар. **назъбени** (jagged масиви).
- Всеки ред на jagged array на практика е масив, който може да има различен брой елементи от останалите в назъбения масив, но не може да има различна размерност.
- Със следващия код декларираме масив от масиви:

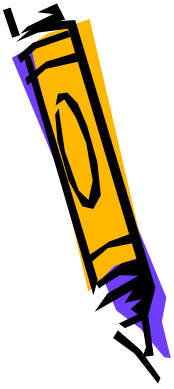
```
int[][] jaggedArray;
```



Масиви от масиви (jagged масиви)

- Единственото по-особено е, че нямаме само една двойка скоби, както при обикновените масиви, а имаме вече две двойки такива.
- По следния начин заделяме назъбен масив:

```
jaggedArray = new int[2][];  
jaggedArray[0] = new int[5];  
jaggedArray[1] = new int[3];
```



Масиви от масиви (jagged масиви)

- Както споменахме, елементите на назъбения масив може и да са не само едномерни масиви, но и многомерни такива.
- В следващия код създаваме назъбен масив от двумерни масиви:

```
int[][,] jaggedOfMulti = new int [2][,];  
jaggedOfMulti[0] = new int[,] { {9,27}, {10,20}};  
jaggedOfMulti[1] = new int[,] { {19,3}, {1,3}};
```



Масиви от масиви (jagged масиви)

- Възможно е и декларирането, заделянето и инициализацията на един масив от масиви да се извършва по следния начин:

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



- Массиви от массиви (jagged массиви)
- Възможно е и декларирането, заделянето и инициализацията на един массив от массиви да се извършва в един израз. Ето примери:

```
int[][] myJaggedArray =  
{  
    new int[] {1,3,5,7,9},  
    new int[] {17,23},  
    new int[] {0,2,4,6}  
};
```



- **Масиви от масиви (jagged масиви)**

//Или с инстанцииране и задаване размера на I-
вия масив:

```
int[][] myJaggedArray = new int [3][]  
{  
    new int[] {1,3,5,7,9},  
    new int[] {17,23},  
    new int[] {0,2,4,6}  
};
```

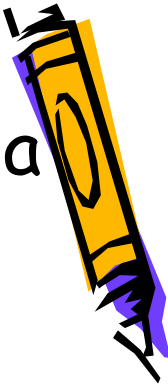


- **Масиви от масиви (jagged масиви)**

//Или с инстанцииране без задаване размера на
I-вия масив:

```
int[][] myJaggedArray = new int [][]  
{  
    new int[] {1,2,5,7,9},  
    new int[] {17,23},  
    new int[] {0,2,4,6}  
};
```

myJaggedArray[0]	1	2	5	7	9
myJaggedArray[1]	17	23			
myJaggedArray[2]	0	2	4	6	



- **Масиви от масиви (jagged масиви)**

myJaggedArray[0]	1	2	5	7	9
myJaggedArray[1]	17	23			
myJaggedArray[2]	0	2	4	6	

- **Достъп до елементите на jaggedArray**
- Достъпът до елементите на масивите, които са част от назъбения, отново е по индекс. Ето пример за достъп до елемента с индекс 3 от масива, който има индекс 0 в по-горе дефинирания назъбен масив **jaggedArray**:

`jaggedArray[0][3] = 33;`

//тоест 5 се променя на 33



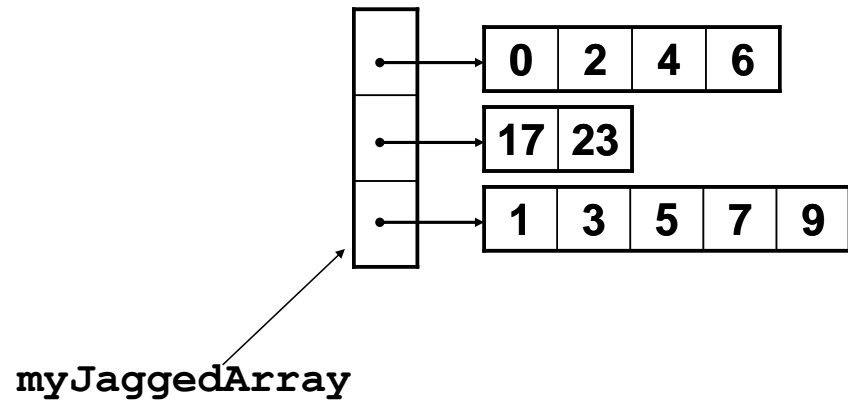
За да се разбере разположението на масива в паметта ще се спрем на следния пример:

```
int[][] myJaggedArray = new int[3][];  
myJaggedArray[0]= new int[] {0,2,4,6};  
myJaggedArray[1]= new int[] {17,23};  
myJaggedArray[2]=new int[] {1,3,5,7,9};
```

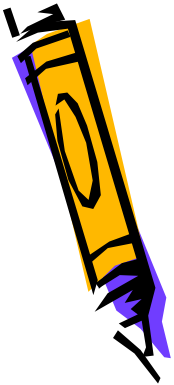
- При деклариране на такъв масив, в частност – масив **myJaggedArray**, в стека се заделя променливата **myJaggedArray**, която сочи към участък от 3 елемента в динамичната памет, които съдържат само референции към масивите (елементи на назъбения масив), а не самите тях, тъй като CLR не знае каква ще е размерността на всеки един от масивите.



- Чак след като се задели памет за някой от масивите (елементи на назъбения масив), тогава референцията се насочва към новосъздадения блок динамична памет.



- Може да видим вече дефинирания назъбен масив **myJaggedArray** и по точно неговото разположение в паметта (heap-a): самата променлива **myJaggedArray** съдържа само референции към масивите, а не самите тях.



Пример - сортиране на масиви (колекции), чиито елементи са от тип, дефиниран от потребителя (в случая – клас).

За да се сортира един такъв масив (колекция), трябва да сме в състояние да сравняваме елементите му. За сравнение се използват интерфейсите **Comparable** и **Comparer**.

```
public interface Comparable  
{int CompareTo(object o);}
```

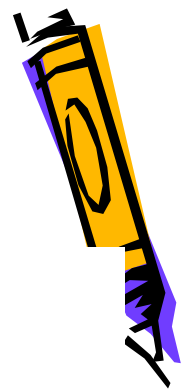
//сравнява 1 текущ обект с обект от същия тип
(подаден като параметър)

//връща стойност, която индикира дали обектите са
равни или не

Ако стойността е <0, то текущия обект е < от
параметъра o

Ако стойността е >0, то текущия обект е > от
параметъра o

Ако стойността е =0, то текущия обект е равен на
параметъра o



```
public interface IComparer  
{int Compare(object x, object y);}
```

//сравнява 2 обекта от различен тип, тоест може да се използва в хетерогенни колекции

//върща стойност, която индикира дали обектите са равни или не

Ако стойността е <0 , то Инстанцията е $<$ от обекта

Ако стойността е >0 , то Инстанцията е $>$ от обекта

Ако стойността е $=0$, то Инстанцията е равна на обекта



using System;

```
class st : IComparable
```

```
{
```

```
    public string ime;
```

```
    public int uspeh;
```

```
    //Конструктор
```

```
    public st(string im, int u)
```

```
    { ime = im; uspeh = u; }
```

```
    public int CompareTo(Object o)
```

```
    //Имплементация на интерфейсен метод
```

```
    {
```

```
//1. Сортиране по успех във възходящ ред
```

```
        st my = (st)o;
```

```
        if (uspeh > my.uspeh) return 1;
```

```
        else if (uspeh == my.uspeh) return 0;
```

```
        else return -1;
```

```
//2. Сортиране по успех във низходящ ред
```

```
        //if (uspeh > my.uspeh) return -1;
```

```
        //else if (uspeh == my.uspeh) return 0;
```

```
        //else return 1;
```

```
//3. Сортиране по име във възходящ ред
```

```
        //int
```

```
d=(String.Compare(ime,my.ime));
```

```
        //if (d>0) return 1;
```

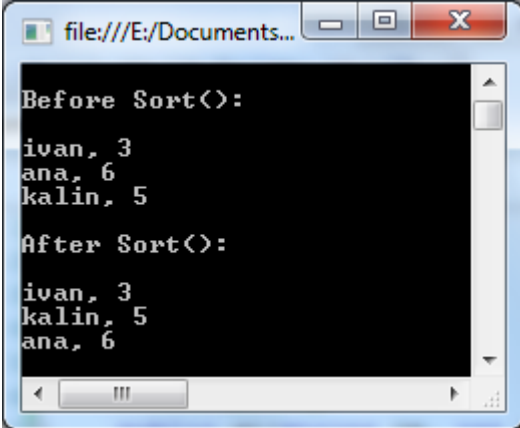
```
        //else if (d == 0) return 0;
```

```
        //else return -1;
```

```
    }
```

```
}
```

Sort - c IComparable



```
file:///E:/Documents...
Before Sort():
ivan, 3
ana, 6
kalin, 5

After Sort():
ivan, 3
kalin, 5
ana, 6
```

```
class Program
```

```
{ static void print(st[] k)
```

```
{
```

```
    foreach (st buf in k)
```

```
    Console.WriteLine("{0}, {1}", buf.ime,
```

```
    buf.uspeh);
```

```
}
```

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

```
{
```

```
    st[] s = new st[3]; //масив
```

```
    s[0] = new st("ivan", 3);
```

```
    s[1] = new st("ana", 6);
```

```
    s[2] = new st("kalin", 5);
```

```
    Console.WriteLine("\nBefore Sort():\n");
```

```
    print(s);
```

```
    Console.WriteLine("\nAfter Sort():\n");
```

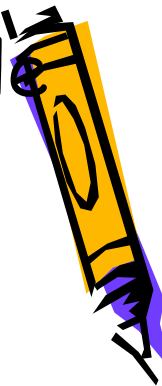
```
    Array.Sort(s);
```

```
    print(s);
```

```
    Console.ReadLine();
```

```
}
```

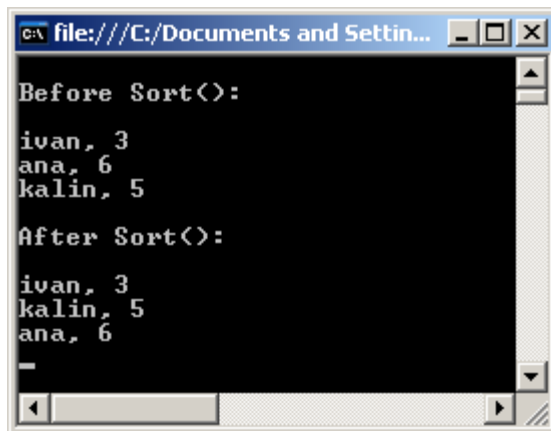
```
}
```



```

using System;
using System.Collections;
class st
{
    public string ime;
    public int uspeh;
    //Конструктор
    public st(string im, int u)
    { ime = im; uspeh = u; }
}
class StudentAgeComparer : IComparer
{
    public int Compare(object aE11, object aE12)
    {
        st student1 = aE11 as st;
        st student2 = aE12 as st;
        return
student1.uspeh.CompareTo(student2.uspeh);
    }
}

```



```

file:///C:/Documents and Settings...
Before Sort():
ivan, 3
ana, 6
kalin, 5

After Sort():
ivan, 3
kalin, 5
ana, 6
-

```

Sort - c IComparer

```

class Program
{
    static void print(st[] k)
    {
        foreach (st buf in k)
            Console.WriteLine("{0}, {1}", buf.ime,
buf.uspeh);
    }
    static void Main(string[] args)
    {
        st[] s = new st[3]; // definiram masivu
        s[0] = new st("ivan", 3);
        s[1] = new st("ana", 6);
        s[2] = new st("kalin", 5);
        Console.WriteLine("\nBefore Sort():\n");
        print(s);
        Console.WriteLine("\nAfter Sort():\n");
        Array.Sort(s, new
StudentAgeComparer()); // Array.Sort(s);
        print(s);
        Console.ReadLine();
    }
}

```

