



# Arrays (Diziler) Collections (Koleksiyonlar)

Adem AKKUŞ

Bilgisayar Mühendisi  
Uzman Bilişim Teknolojileri Öğrt.  
Eğitmen

<b>Collections</b>	<b>Generic</b>
Array ArrayList	List<T>
HashTable SortedList	SortedList<T> Dictionary<T>
Queue Stack	Queue<T> Stack<T>



# Arrays (Diziler)

- Dizi, aynı tipte birden çok değeri bellek üzerinde tutabilecek yapıdır.
- Programlama yaparken dizileri kullanmak; dizilerin verdiği avantajlardan yararlanarak değerler üzerinde;

- ☐ seçme,

- ☐ silme,

- ☐ değiştirme,

- ☐ sıralama

vb. işlemlerin kolayca gerçekleştirilmesini sağlar.

# Arrays (Diziler)

- C# dilinde tanımlanan bütün diziler **System.Array** namespace'i altında yer almaktadır.
- Aynı türden birden fazla değeri tek bir referans altında saklamak için kullanılan yapılardır.

`byte[] sayilar={10,20,30,40,50};`

adres	değer	} sayilar
100h	10	
101h	20	
102h	30	
103h	40	
104h	50	

- Diziler referans tipli olduğu için belleğin **heap** alanında saklanırlar.

# Dizi Tanımlamak

Dizi oluştururken temelde üç noktaya dikkat edilir.

**Dizinin Tipi:** Dizide hangi tip verilerin saklanacağı (int, string, char, byte, double vb.) belirtilir.

**Dizi Adı:** Dizide saklanacak verilerle anlamlandırılan değişken isimlendirme kurallarına uygun hangi isimlerin diziye verileceği belirtilir (Anlamlı isimler vermek, yazılan kodların okunabilirliğini artıracak için her zaman tavsiye edilen bir yöntemdir. Çoğul eki (-ler/-lar) kullanılabilir. Örnek sayılar, isimler, şehirler, dersler gibi

Örneğin okul numaraları saklanacak bir dizi için diziOkulNo gibi

```
int[] sayilar = new int[10];
```

Dizinin Tipi  
Dizi Adı  
Dizi Kapasitesi

# Dizi Tanımlamak

**Dizilerin Kapasitesi:** Dizi içinde kaç adet veri saklanacağı belirtilir. sayılar isminde, **integer** tipinde **10** adet veri saklama kapasitesine sahip bir dizi tanımlaması yapılmıştır. Derleyicinin bir dizi tanıması için başlangıçta veri tipi belirtildikten sonra içi boş köşeli parantezler kullanılmalıdır. İçi boş köşeli parantezler, bu ifadenin bir boyutlu dizi olduğu anlamına gelir. Tanımlamadaki ikinci köşeli parantez ise dizide saklanacak değer sayısını belirtir. Aşağıda farklı veri tiplerine sahip dizi tanımlama örnekleri verilmiştir

```
string[] isimler = new string[5]; // String tipinde 5 elemanlı dizidir.  
byte[] siralar = new byte[6];    // Byte tipinde 6 elemanlı dizidir.  
bool[] durumlar = new bool[4];   // Boolean tipinde 4 elemanlı dizidir.  
float[] uzunluklar = new float[8]; // Float tipinde 8 elemanlı dizidir.
```



# Dizi Tanımlamak

Bir dizi tanımlaması yapıldığında derleyici, dizinin her elemanına temel veri tipleri için varsayılan değerleri ilk değer olarak verir. İlk değerler, dizi içine veri eklenmeden verilir.

```
string[] isimler = new string[5]; // String tipinde 5 elemanlı dizidir.  
byte[] siralar = new byte[6];    // Byte tipinde 6 elemanlı dizidir.  
bool[] durumlar = new bool[4];   // Boolean tipinde 4 elemanlı dizidir.  
float[] uzunluklar = new float[8]; // Float tipinde 8 elemanlı dizidir.
```

Bunlar;

string tipi için null,

**sayısal** tipler (byte,short,int,long,float,double,decimal) için **0**,

**bool** tipi için ise **false**

değerleridir. Verilen bu ilk değerler, dizilere değer aktarımı yapıldıkça yeni değerlerle değiştirilir.

# Dizi Tanımlamak

**veritipi[ ]** diziAdi;

`byte[ ] sayilar=new byte[2];` (Değerler önceden bilinmiyorsa)

`sayilar[0]=1;`

`sayilar[1]=2;`

`byte[ ] sayilar=new byte[ ] { 1 , 2 };` (Değerler önceden belli)

`byte[] sayilar={ 1 , 2 };` (Değerler önceden belli)



# Farklı Türden Diziler

```
char[ ] karakterler=new char[ ]  
{  
    'A','2','B',(char)68  
}
```

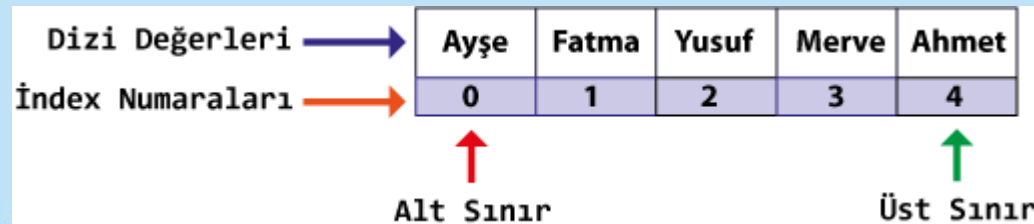
```
Console.WriteLine(karakterler[1]);
```

# Bir Boyutlu Dizi Değer Atama

Dizilere değer aktarmanın farklı yöntemleri vardır. Dizilere ilk olarak tanımlamasının yapıldığı satırda değer verilebilir.

```
string[] kisiler = new string[5] {"Ayşe","Fatma","Yusuf","Merve","Ahmet"};
```

Tanımlanan ve aynı satırda değer aktarımı yapılan dizide kodlar derlendiğinde bellekte 5 elemanlı bir dizi oluşturulur. Oluşturulan bu diziye küme parantezi { } içindeki değerler sırasıyla verilir



Not: dizi ismi aynı zamanda dizinin ilk elemanının adresini gösterir.

# Dizi Tanımlamak

Günlük hayatta sıralama işlemlerine hep 1'den başlanır fakat programlama dillerinin çoğunda sıralama 0'dan başlar. Aşağıdaki görselde görüldüğü üzere eklenen ilk elemanın sıra numarası 0'dır. Dizilerin her değerinin bir sıra numarası vardır. Sıra numaraları index, indis veya indeks olarak adlandırılır.

Dizi Değerleri	→	Ayşe	Fatma	Yusuf	Merve	Ahmet
Index Numaraları	→	0	1	2	3	4
		↑				↑
		Alt Sınır				Üst Sınır

**Not:** Fortran programlama dilinde index değeri 1'den ile başlar.



# Dizi Tanımlamak

Tanımlandıkları satırda dizilere değer aktarma işlemi farklı şekillerde de yapılabilir.

Aşağıdaki örnekte diziye değer aktarım işlemi, dizinin eleman sayısı belirtilmeden veya **new** anahtar sözcüğü kullanılmadan gerçekleştirilmiştir. Bu durumda derleyici hata vermez ve dizinin eleman sayısı derleyici tarafından belirlenir.

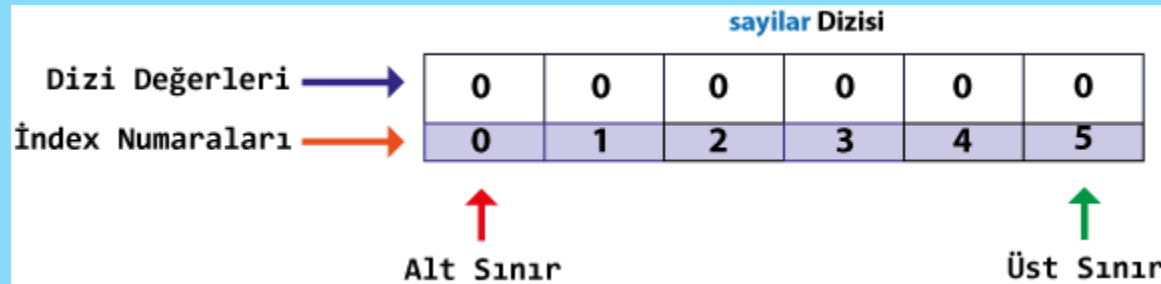
```
string[] kisiler = new string[] {"Ayşe","Fatma","Yusuf","Merve","Mehmet"};  
string[] kisiler = {"Ayşe","Fatma","Yusuf","Merve","Mehmet"};
```

Dizilere değer aktarımının bir diğer yöntemi, dizinin **index** numaralarının kullanılarak yapılmasıdır.

```
int[] sayilar = new int[6];
```

# Dizi Tanımlamak

Kodda integer veri tipine sahip, 6 elemanlı, **sayilar** adında bir dizi tanımlandı. Derleyici, bu kod satırına geldiğinde bellekte eleman sayısı kadar yer ayırır ve bu yerlere ilk değer olarak **0** (sıfır) sayısını aktarır.



Bellek üzerinde dizi oluşturulduktan sonra index numaraları kullanılarak değer aktarımı gerçekleştirilebilir. Görsel **sayilar** ismindeki dizinin 2 numaralı **index** elemanına (dizinin üçüncü elemanına) 45 değerinin aktarımı yapılmıştır.



# Dizi Tanımlamak

```
int[] sayilar = new int[6];
```

```
sayilar[0] = 10; //sayilar dizisinin 0 index numaralı elemanı 10 oldu.
```

```
sayilar[1] = 25; //sayilar dizisinin 1 index numaralı elemanı 25 oldu.
```

```
sayilar[2] = 45; //sayilar dizisinin 2 index numaralı elemanı 45 oldu.
```

```
sayilar[3] = 5; //sayilar dizisinin 3 index numaralı elemanı 5 oldu.
```

```
sayilar[4] = -30; //sayilar dizisinin 4 index numaralı elemanı -30 oldu.
```

```
sayilar[5] = -50; //sayilar dizisinin 5 index numaralı elemanı -50 oldu.
```

		sayilar Dizisi					
Dizi Değerleri	→	10	25	45	5	-30	-50
İndex Numaraları	→	0	1	2	3	4	5
		↑					↑
		Alt Sınır					Üst Sınır

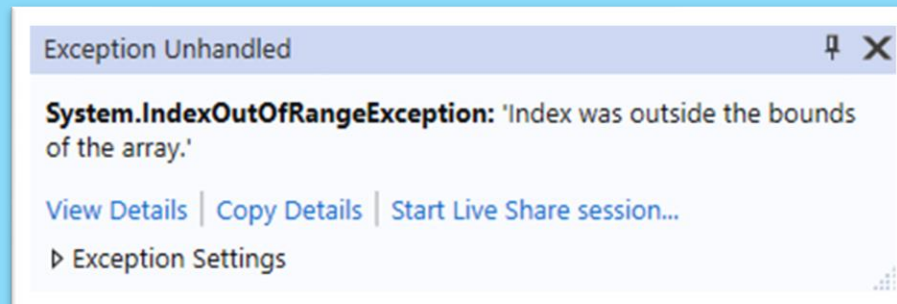


# Dizi Tanımlamak

Değer aktarım işleminde index numaralarına göre dizinin index numarası 0'dan başlayarak dizideki eleman sayısının bir eksiğine kadar istenilen alana değer aktarılabilir. Tanımlanan dizide 0'dan küçük ve eleman sayısının bir eksiğinden büyük bir index numarası ile diziye değer aktarmaya veya dizi elemanına erişmeye çalışıldığında derleyici tarafından hata mesajı gönderilir. Hata mesajı, girilen index numarasının dizinin sınırları dışında olduğunu bildirir.

```
isayilar[6] = -30;
```

Kod yazıldığında derleyici, aşağıdaki hata mesajını vererek kullanıcıyı uyarır.



# Çok Boyutlu Diziler

- Bu dizilerde satır ve sütun belirtilerek veri saklanmaktadır.
- Örneğin aşağıdaki verileri saklamak için çok boyutlu bir dizi kullanılabilir.

Ad	Soyad	Numara	Doğum Tarihi
Hasan	Uçar	30	1991
Metin	Çelik	25	1990

```
string[ , ] ogrenciler=new string[2,4];  
ogrenciler[ 0,0 ]="Hasan";  
Ogrenciler[1, 2 ]="25";
```

# Çok boyutlu Diziler

Ad	Soyad	Numara	Doğum Tarihi
Hasan	Uçar	30	1991
Metin	Çelik	25	1990

string[,] ogrenciler =

{

    {"Hasan" , "Uçar", "30", "1991"},

    {"Metin", "Çelik", "25", "1990"},

};



# Çok boyutlu Diziler

Ad	Soyad	Numara	Doğum Tarihi
Hasan	Uçar	30	1991
Metin	Çelik	25	1990

```
int rowCount=ogrenciler.GetLength(0);
```

```
int colCount=ogrenciler.GetLength(1);
```

```
rowCount=2
```

```
colCount=4 olacaktır.
```

# CreateInstance Metodu İle Dizi

Array.CreateInstance yöntemi ile oluşturulan dizilerin derleme zamanında türünün bildirilmesi gerekmez. Tek boyutlu ve çok boyutlu diziler oluşturulabilir.

*Array.CreateInstance(elementtype, length) ;*  
söz dizimine sahiptir.

CreateInstance(Type, Int32)

CreateInstance(Type, Int32[])

CreateInstance(Type, Int64[])

CreateInstance(Type, Int32, Int32)

CreateInstance(Type, Int32[], Int32[])

CreateInstance(Type, Int32, Int32, Int32)

# CreateInstance Metodu İle Dizi

Array.CreateInstance yöntemi ile oluşturulan dizilerin derleme zamanında türünün bildirilmesi gerekmez. Tek boyutlu ve çok boyutlu diziler oluşturulabilir.

*Array.CreateInstance(elementtype, length) ;*  
söz dizimine sahiptir.

```
public static void Main(string[] args)
{
    Array array = Array.CreateInstance(typeof(int), 5);
    int[] values = (int[])array;
    Console.WriteLine(values.Length);

    Console.ReadLine();
}
```



# CreateInstance Metodu İle Dizi

Array.CreateInstance yöntemi ile oluşturulan dizilerin derleme zamanında türünün bildirilmesi gerekmez. Tek boyutlu ve çok boyutlu diziler oluşturulabilir.

```
public static void Main() {  
  
    // Creates and initializes a multidimensional Array of type string.  
    int[] myLengthsArray = new int[4] { 2, 3, 4, 5 };  
    Array my4DArray=Array.CreateInstance( typeof(string), myLengthsArray );  
    for ( int i = my4DArray.GetLowerBound(0); i <= my4DArray.GetUpperBound(0); i++ )  
        for ( int j = my4DArray.GetLowerBound(1); j <= my4DArray.GetUpperBound(1); j++ )  
            for ( int k = my4DArray.GetLowerBound(2); k <= my4DArray.GetUpperBound(2); k++ )  
                for ( int l = my4DArray.GetLowerBound(3); l <= my4DArray.GetUpperBound(3); l++ ) {  
                    int[] myIndicesArray = new int[4] { i, j, k, l };  
                    my4DArray.SetValue( Convert.ToString(i) + j + k + l, myIndicesArray );  
                }  
  
    // Displays the values of the Array.  
    Console.WriteLine( "The four-dimensional Array contains the following values:" );  
    PrintValues( my4DArray );  
}
```

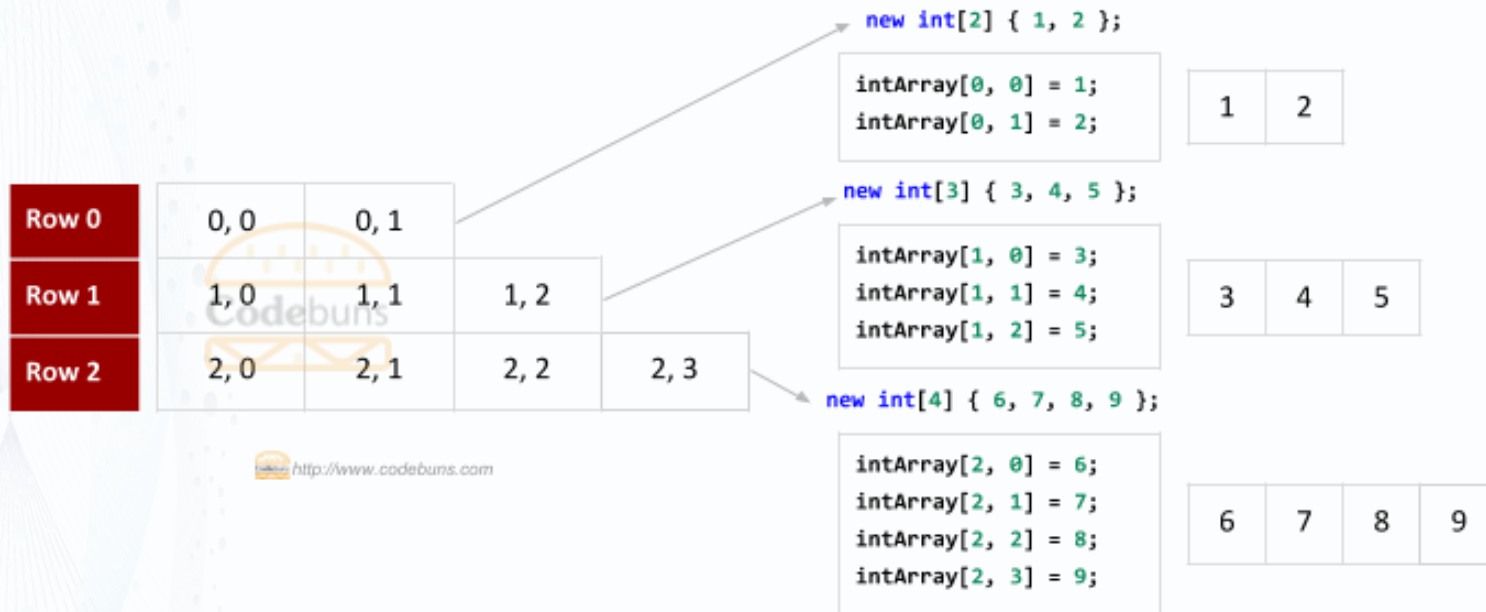
# CreateInstance Metodu İle Dizi

Array.CreateInstance yöntemi ile oluşturulan dizilerin derleme zamanında türünün bildirilmesi gerekmez. Tek boyutlu ve çok boyutlu diziler oluşturulabilir.

```
public static void PrintValues( Array myArr ) {  
    System.Collections.IEnumerator myEnumerator = myArr.GetEnumerator();  
    int i = 0;  
    int cols = myArr.GetLength( myArr.Rank - 1 );  
    while ( myEnumerator.MoveNext() ) {  
        if ( i < cols ) {  
            i++;  
        } else {  
            Console.WriteLine();  
            i = 1;  
        }  
        Console.Write( "\t{0}", myEnumerator.Current );  
    }  
    Console.WriteLine();  
}
```

# Jagged Arrays (Düzensiz Diziler)

İç içe geçmiş diziler, dizi içeren dizilerdir. Tek boyutlu dizinin elemanı olarak dizi verildiğinde oluşur. Çok fazla kullanılmaz çünkü iç içe diziler performans açısından çok da iyi bir seçenek değildir. Örnek kullanımına bakalım.





# Jagged Arrays (Düzensiz Diziler)

İç içe geçmiş diziler, dizi içeren dizilerdir. Tek boyutlu dizinin elemanı olarak dizi verildiğinde oluşur. Çok fazla kullanılmaz çünkü iç içe diziler performans açısından çok da iyi bir seçenek değildir. Örnek kullanımına bakalım.

```
int[][] arr = new int[2][];
```

```
arr[0] = new int[4];  
arr[1] = new int[6];
```

```
arr[0] = new int[4] { 11, 21, 56, 78 };  
arr[1] = new int[6] { 42, 61, 37, 41, 59, 63 };
```

```
arr[0] = new int[] { 11, 21, 56, 78 };  
arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };
```

# Jagged Arrays (Düzensiz Diziler)

İç içe geçmiş diziler, dizi içeren dizilerdir. Tek boyutlu dizinin elemanı olarak dizi verildiğinde oluşur. Çok fazla kullanılmaz çünkü iç içe diziler performans açısından çok da iyi bir seçenek değildir. Örnek kullanımına bakalım.

```
int[][, ] jagged_arr1 = new int[4][, ]  
{  
    new int[, ] { {1, 3}, {5, 7} },  
    new int[, ] { {0, 2}, {4, 6}, {8, 10} },  
    new int[, ] { {7, 8}, {3, 1}, {0, 6} },  
    new int[, ] { {11, 22}, {99, 88}, {0, 9} }  
};
```

# Jagged Arrays (Düzensiz Diziler)

İç içe geçmiş diziler, dizi içeren dizilerdir. Tek boyutlu dizinin elemanı olarak dizi verildiğinde oluşur. Çok fazla kullanılmaz çünkü iç içe diziler performans açısından çok da iyi bir seçenek değildir.

```
int[][] arr = new int[2][]; // Declare the array

arr[0] = new int[] { 11, 21, 56, 78 }; // Initialize the array
arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };

// Traverse array elements
for (int i = 0; i < arr.Length; i++)
{
    for (int j = 0; j < arr[i].Length; j++)
    {
        System.Console.Write(arr[i][j] + " ");
    }
    System.Console.WriteLine();
}
```

Output:

```
11 21 56 78
42 61 37 41 59 63
```



# Dizilerde foreach Döngüsü

Birçok programlama dili, diziler üzerinde işlem yapılmasını kolaylaştıran bir döngü sunar.

Bu döngü, **foreach** döngüsüdür. Dizilerde kullanılan foreach, dizi elemanlarını ilk elemandan başlayıp, dizinin son elemanına kadar her elemanı tek tek dolaşarak belirlenen bir değişkene aktarır.

Örneğin 10 elemanlı bir dizide foreach döngüsü kullanıldığında döngü 10 defa tekrarlama işlemi yapar. Döngü her seferinde dizi içindeki değeri alarak aynı tipte olan bir değişkene aktarır. Döngünün yapısı aşağıda verilmiştir.

```
foreach (Tip Değişken in Dizi)
{
    // Döngü içindeki işlemler
}
```

# Dizilerde foreach Döngüsü

**foreach** döngüsünün yapısındaki ögeler aşağıda sıralanmıştır.

**Tip:** Dizi içindeki veri tipleri ile aynı olmalıdır (Dizi içindeki değerler string ise Tip de string, double ise Tip de double olmalıdır.). Bazı durumlarda Tip olarak **var** kullanılır. Var tipi, kendisine atanan değer ne ise o değer tipini alır.

**Değişken:** Döngü, dizi içindeki değeri her dönme işleminde belirtilen bir değişkene aktarır.

**in:** Bir anahtar kelimedir, foreach döngülerinde değişken adlarından sonra kullanılır.

**Dizi:** Üzerinde işlem yapılacak dizinin adıdır.

```
int[] sayilar = { 20, 30, 40, 50 };  
foreach (int sayi in sayilar)  
{  
    Console.WriteLine(sayi);  
}
```

```
int[] sayilar = { 20, 30, 40, 50 };  
for (int i = 0; i < sayilar.Length; i++)  
{  
    Console.WriteLine(sayilar[i]);  
}
```

# Koleksiyon Sınıfları

Klâsik programlama dillerinde array çok önemli bir veri tipidir. Çok sayıda değişkeni kolayca tanımlar ve o değişkenlere istemli (random) erişim sağlar.

Ancak array tipinin iki önemli handikapı vardır:

- 1.Array 'in öğeleri aynı veri tipinden olmalıdır,
- 2.Array 'in boyutu (öğesi sayısı) önceden belli edilmelidir.



# Koleksiyon Sınıfları

Oysa, programlama işinde, çoğunlukla aynı veri tipinden olmayan topluluklarla karşılaşırız.

C# bu tür toplulukları ele alabilmek için, array yapısından çok daha genel olan koleksiyon (**collection**) veri tipini getirmiştir.

Koleksiyon veri tipi, array veri tipinin çok kullanışlı bazı özelliklerini herhangi bir nesneler topluluğuna taşıma olanağı sağlamıştır.

# Koleksiyon Sınıfları

Koleksiyon sınıfları nesnelerden oluşan topluluklardır. C#,

- koleksiyonları oluşturmak,
- koleksiyona yeni öge ekleme,
- koleksiyondan öge atmak,
- koleksiyonun öğelerini sıralamak, numaralamak,
- koleksiyon içinde öge aramak

vb işleri yapmamızı sağlayan sınıflar, metotlar ve arayüzlerden oluşan çok geniş bir kütüphaneye sahiptir.

# Koleksiyon Sınıfları

Ayrıca, kullanıcı kendi koleksiyonunu oluşturabilir, onlarla istediği işi yapmayı sağlayacak metotları ve arayüzleri oluşturabilir.

C# dilinde koleksiyonlar ad uzayı(namespace) içinde birer veri tipidir.

Klasik dillerdeki array yapısının çok daha gelişmiş biçimleridir.

Not: Python programlama dilinde dizi `array` yoktur. Bunun yerine koleksiyonu karşılığı olan `list` tipi vardır.

# Collections (Non Generic)

- Dizilerde her ne kadar aynı türden birden fazla veri saklanabilmesine rağmen boyutunun (eleman sayısının) tanımlama aşamasında belirtilmesi kısıtlayıcı bir faktördür.

```
byte[] sayilar=new byte[10];
```

- Generic olmayan koleksiyonlarda saklanan tüm elemanlar **object** türünden saklandığından dolayı her türden üyeyi saklayabilmektedirler.

```
ArrayList sayilar=new ArrayList();
```

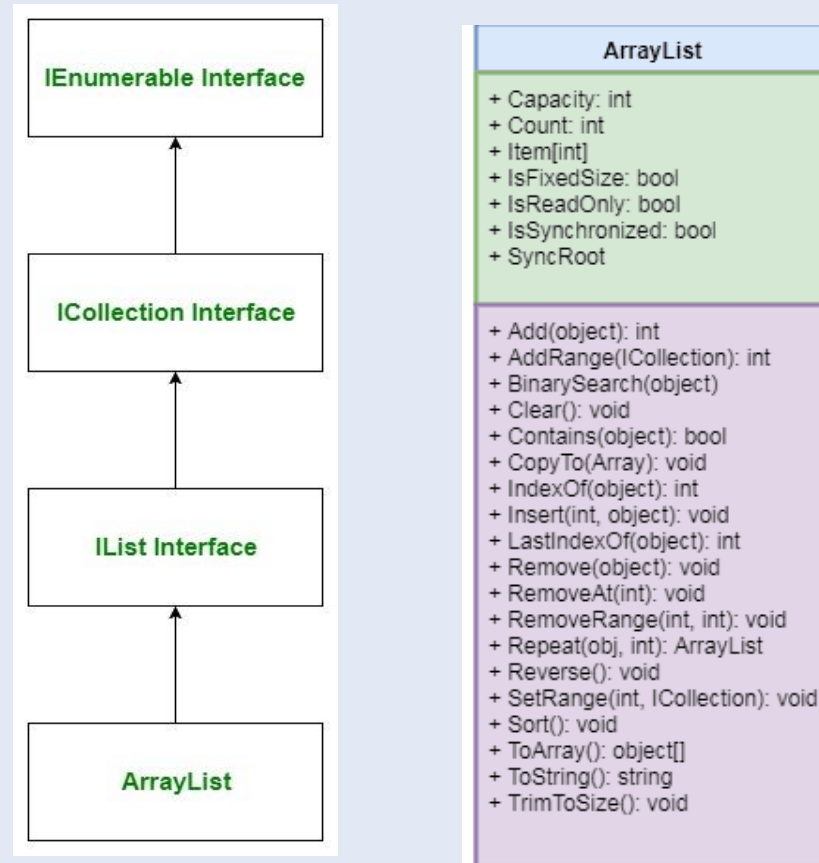


# Collections (Non Generic)

- Ancak saklanan üye kendi türünü kaybederek object türüne dönüştürülerek saklandığından dolayı (**boxing**) ilgili üye kullanılmak istendiğinde tekrar kendi türüne dönüştürülmelidir (**unboxing**).

```
ArrayList sayilar=new ArrayList();  
sayilar.Add(20); //object türünde  
sayilar.Add(30); //object türünde  
int deger=Convert.ToInt32 (sayilar[0] ) //unboxing
```

# Collections (Non Generic)



# ArrayList

```
ArrayList sayilar=new ArrayList();  
sayilar.Add(2);  
sayilar.Add(3);
```

```
int toplam= sayilar[0]    +    sayilar[1];  
                (object)          (object)
```

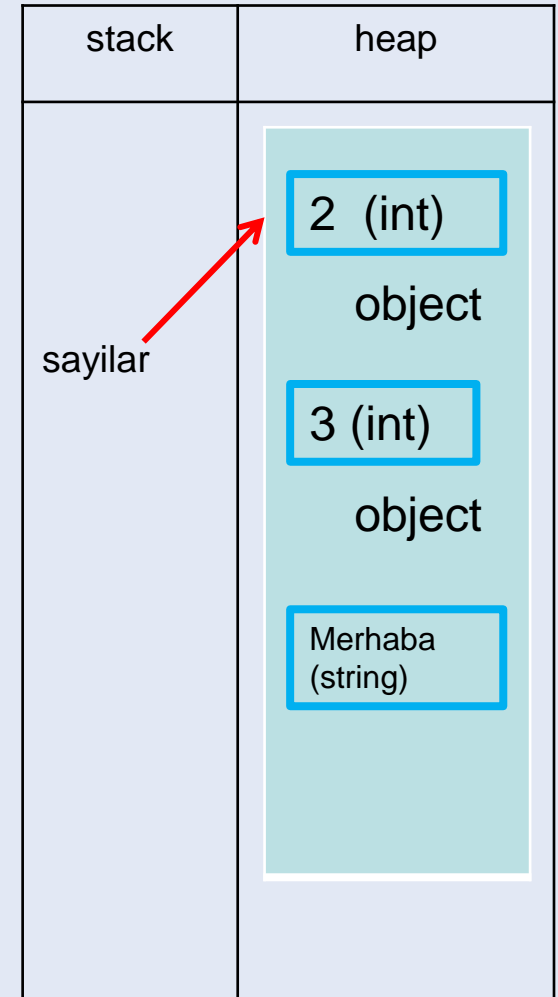
**object + object =int**                      değildir!

# ArrayList

```
ArrayList sayilar=new ArrayList();  
sayilar.Add(2);  
sayilar.Add(3);  
sayilar.Add("Merhaba");
```

```
int toplam=Convert.ToInt32 (sayilar[0] ) +  
Convert.ToInt32 (sayilar[1] ) ;
```

```
foreach (int sayi in sayilar)  
{  
    Console.WriteLine(sayi.ToString());  
}
```





# ArrayList Metotları

- **Sort metodu**

Koleksiyon üyelerini sıralamak için kullanılır.

dizi.Sort();

```
using System;
using System.Collections;
namespace Koleksiyonlar
{
    class Dizi
    {
        static void Main(string[] args){

            ArrayList birDizi = new
            ArrayList();
            birDizi.Add("Zonguldak");
            birDizi.Add("Urfa");
            birDizi.Add("Adana"); birDizi.Add("Bursa"); birDizi.Add("Izmir");
            Console.WriteLine("Sıralanmamış Liste"); foreach (string s in
            birDizi)
            Console.WriteLine(s.ToString()); Console.WriteLine();
            Console.WriteLine("Sıralanmış Liste");
            birDizi.Sort();
            foreach (string s in birDizi) Console.WriteLine(s.ToString());
            }}}}
```

# ArrayList

```
ArrayList dizi=new ArrayList();  
dizi.Add(2); dizi.Add(1); dizi.Add(3);
```

- **Sort metodu**

Koleksiyon üyelerini sıralamak için kullanılır.

```
dizi.Sort();
```

- **Count özelliği**

Koleksiyonun eleman sayısını verir.

# ArrayList

```
ArrayList dizi=new ArrayList();  
dizi.Add(2); dizi.Add(1); dizi.Add(3);
```

- **Remove metodu**

Koleksiyon üyelerini sıralamak için kullanılır.  
`dizi.Remove(2);` Silinecek üye belirtilir.

- **RemoveAt metodu**

Koleksiyondan index değeri belirtilen elemanı siler. Elemanları da silinen elemana doğru kaydırır.

```
dizi.RemoveAt(1);
```

# Hashtable

Hashtable sınıfı, Object tiplerin hızla depolanması ve depodan hızla çekilmesi için iyi yöntemleri olan bir yapıdır.

Anahtarlara dayalı arama yapar. Anahtarlar belli tiplere oluşturulmuş hash kodlardan ibarettir.

GetHashCode() metodu yaratılan bir nesnenin hash kodunu verir. Aşağıdaki program parçası Hashtable sınıfının nasıl kullanıldığını göstermektedir.



# Hashtable

- Değerlerin index numarası yerine anahtar yardımıyla saklandığı koleksiyon türüdür.

```
Hashtable ht = new Hashtable();
```

```
ht.Add(1, "Osman");
```

```
ht.Add(2, "Hasan");
```

key

value

```
ht.Add(2, "Murat");
```

Aynı key değerine sahip eleman eklenemez.

# Hashtable

```
Hashtable ht = new Hashtable();
```

```
ht.Add(1, "Osman");
```

```
ht.Add(2, "Hasan");
```

```
ICollection keys = ht.Keys;
```

```
ICollection values = ht.Values;
```

```
foreach (object item in values)
```

```
{
```

```
    Console.WriteLine(item);
```

```
}
```

# HashTable

```
using System;
using System.Collections;

class Test
{
    static void Main()
    {
        Hashtable hashTable = new Hashtable();
        hashTable.Add(1, "Gökova"); hashTable.Add(2,
        "Belek"); hashTable.Add(3, "Çamdibi");
        hashTable.Add(4, "Marmaris");
        Console.WriteLine("Anahtarlar:--");
        foreach (int k in hashTable.Keys)
        {
            Console.WriteLine(k);
        }

        Console.WriteLine("Aramak için anahtarı giriniz :");
        int n = int.Parse(Console.ReadLine());
        Console.WriteLine(hashTable[n].ToString());
    }
}
```

# HashTable

Hashtable objelerini listelemek için, yukarıdaki listeleme yöntemi yerine, IDictionaryEnumerator 'i kullanarak aşağıda gösterildiği gibi de yapabiliriz.

```
Hashtable hashTable = new Hashtable(); hashTable.Add(1,
"Matematik"); hashTable.Add(2, "Fizik"); hashTable.Add(3,
"Kimya"); hashTable.Add(4, "Biyoloji"); hashTable.Add(5,
"Bilgisayar"); hashTable.Add(6, "Jeoloji");
Console.WriteLine("Anahtarlar:--");
IDictionaryEnumerator en = hashTable.GetEnumerator();
string str = String.Empty;
while (en.MoveNext())
{
    str = en.Value.ToString();
    Console.WriteLine(str);
}
```



# SortedList

- SortedList sınıfı System.Object tiplerini anahtar-değer çiftine göre yerleştirir; ayrıca sıralama yapar.
- Elemanlara erişimin hem index numarası hem de anahtar yardımıyla yapılabildiği koleksiyondur.

```
SortedList sl = new SortedList();
```

```
sl.Add(1, "Osman");
```

```
sl.Add(2, "Hasan");
```

```
sl.IndexOfKey(1);
```

```
sl.IndexOfValue("Hasan");
```

```
sl.GetKey(0);
```

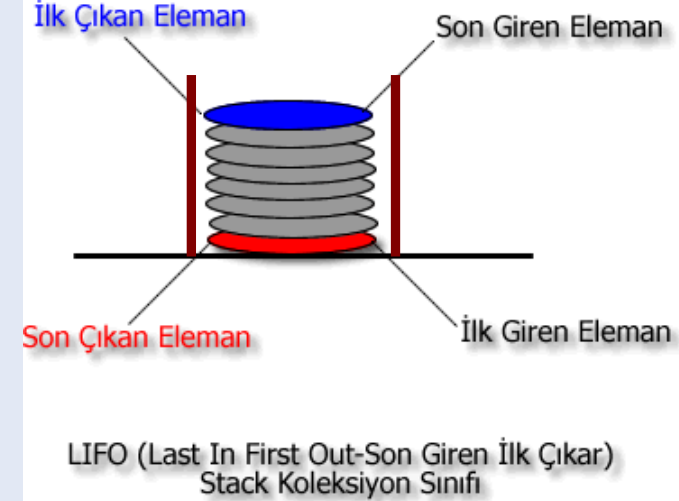
# SortedList

```
using System;
using System.Collections;
using System.Collections.Specialized;

class Test
{
    static void Main()
    {
        SortedList sortedList = new SortedList();
        sortedList.Add(1, "Matematik");
        sortedList.Add(2, "Fizik");
        sortedList.Add(3, "Kimya");
        sortedList.Add(4, "Biyoloji");
        sortedList.Add(5, "Bilgisayar");
        sortedList.Add(6, "Jeoloji");

        Console.WriteLine("Listeyi giriliş sırasıyla yazar:");
        foreach (string str in sortedList.Values)
        {
            Console.WriteLine(str);
        }
    }
}
```

# Stack



- Son elemana ulaşmak en kolaydır. İlk elemana ulaşmak için diğer bütün elemanları çıkarmak gerekir.
- **Push()** metodu koleksiyona eleman eklemek için kullanılır.
- **Pop()** metodu son giren elemanı verirken bu elemanı koleksiyondan siler.

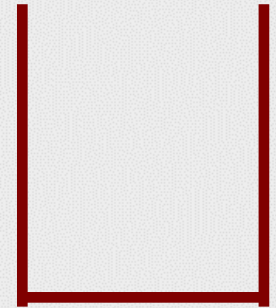
Bunun önüne geçen metod Peek() metodudur.



120

True

"deneme"



```
Stack stc=new Stack();
```

```
Stc.Push("deneme");
```

```
Stc.Push(120);
```

```
Stc.Push(true);
```

```
Stc.Pop();
```

```
Stc.Peek();
```

120



# Collections

## Queue :

**Enqueue()** metodu koleksiyona eleman eklemek için kullanılır.

**Dequeue()** metodu koleksiyona giren elemanı verirken bu elemanı koleksiyondan siler.

Bunun önüne geçen metod **Peek()** metodudur.

# Collections

## Queue:

```
Queue siras=new Queue();  
Stc.Enqueue("deneme");  
Stc.Enqueue(120);  
Stc.Enqueue(true);  
Stc.Dequeue();  
Stc.Peek();
```

deneme

120

true

120



# Generic Koleksiyonlar

## List<T>:

En verimli çalışan ve en çok kullanılan Generic sınıfımızdır. ArrayList sınıfının Generic versiyonudur.

# Generic Koleksiyonlar

**List<T> :**



**List<TextBox> lst = new List<TextBox>();**

**TextBox txt1=new TextBox();**

**TextBox txt2=new TextBox();**

**Lst.Add(txt1);**

**Lst.Add(txt2);**



# Generic Koleksiyonlar

## Dictionary<> ve SortedList<>:

HashTable ve SortedList yapısını kullanmaktadırlar. Tek fark Key ve Value değerlerinin generic olmasıdır. Bu da her bir veri için iki adet boxing işleminden kurtulmak demektir. Bu da bize çok fazla performans artışı sağlar.

# Generic Koleksiyonlar

Dictionary<TKey,TValue>

SortedList<TKey,TValue>



```
Dictionary<int, string> d = new  
Dictionary<int, string>();
```

```
d.Add(1,"metin1");
```

```
d.Add(2,"metin2");
```

```
d.Values----- Verileri dizi şeklinde getirir.
```



# Generic Koleksiyonlar

Stack<T>

Queue<T>




```
Stack<int> stc = new Stack<int>();  
stc.Push(5);  
stc.Push(10);  
stc.Pop();  
stc.Peek();
```

# Generic Koleksiyonlar

Stack<T>

Queue<T>



```
Queue<int> q = new Queue<int>();  
stc.Enqueue(5);  
stc.Enqueue(10);  
stc.Dequeue();  
stc.Peek();
```