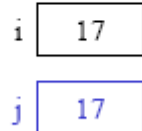
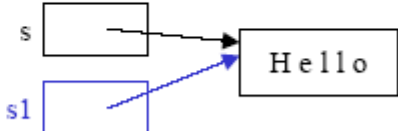


Değer tipleri Referans Tiplerine Karşı

	Value Types	Reference Types
variable contains	value	reference
stored on	stack	heap
initialisation	0, false, '\0'	null
assignment	copies the value	copies the reference
example	<pre>int i = 17; int j = i;</pre> 	<pre>string s = "Hello"; string s1 = s;</pre> 

(Geçen hafta)

Operatörler

- *Operatör sınıflandırılması.*
- *Operatör türleri*

Operatörler

- *Operatör sınıflandırılması (yapılarına göre)*
 - *Unary operatörler*
 - ++, --, new, !, ~
 - *Binary Operatörler*
 - *, /, %, +, =, >>, <<, &&, ^, +=
 - *Ternary(üçlü) Operatörler*
 - ? :

Operatörler

- *Operatör sınıflandırılması (işlevlerine göre)*
 - *Aritmetik operatörler*
 - *Karşılaştırma*
 - *Mantıksal*
 - *Bitsel*
 - *Atama ve işlemlili atama*
 - *Özel amaçlı*

Operatörler

- **Operatör sınıflandırılması (işlevlerine göre)**
 - **Aritmetik operatörler** (++/--,*/,+,-,%) *a=5 Write(++a)*
a=6 , a=6
 - **Karşılaştırma**(<,>,<!=, ==,**as, is**)
 - **Mantık**(||,&&, !) *x>10 || (il="Kocaeli« && max==10)*
 - **Bitisel** (>>,<<,<~(değil),&,<|,<^(xor-aynı 0,farklı 1))
 - **Atama ve işlemlili atama**(*=,%=-= ,+=vs) *a+=5 a=a+5*
 - **Özel amaçlı** (? : operatörü (*a<b ? min=a : min=b*))
 - *new operatörü :yeni bir nesne oluşturur.*
 - *typeof : nesnelerin GetType() metodu gibi. Bağlı olduğu System türünü verir.*

Akış kontrol mekanizmaları

- ***Koşul ifadeleri***
 - *if deyimi*
 - *Switch-case deyimi*
- ***Döngü yapıları (loops)***
 - *for döngüsü*
 - *while döngüsü, do-while*
 - *foreach döngüsü*
- ***Atlama(jump)***
 - *break, continue*

Akış kontrol mekanizmaları(if,switch)

- *if (not<0 || not>100)*
{
Console.WriteLine("yanlış not girişi...");
}
- *switch(not)*
{
case "E": topla(a,b);break;
case 2: carp(a,b); break;
default: Console.WriteLine("yanlış giriş...");break;
}

Akış kontrol mekanizmaları(for,while)

*int i=1; // çift sayıları bulan
döngü.*

for (int i=0; i<20; i++)

{

if (sayi%2==0)

{

cift++;

break;

}

}

while(i<20)

{

i++

}

Atlama (jump)

```
i=1
while(i<20) // ilk çift sayıda döngüden çıkacaktır.
{
    if (i%2==0)
    {
        cift++;
        break;
    }
    i++;
}
```

Akış kontrol mekanizmaları(foreach)

- `string[] isimler = { "Ozlem", "Nesrin", "Ozge", "Fulya" };`
`//string dizi tanımı`

```
foreach (string ss in isimler) // elemanlar yazdırılıyor
{
    Console.WriteLine(ss);
    Ss=yeni değer //yasak
}
```

- `foreach (object i in dizi) // i sadece okunur(readonly)`
`{ Console.WriteLine(i);`
`x=i+1;`
`i++; //yasaklı`

Diziler(System.Array)

- Aynı veri tiplerinin oluşturduğu veri yapısıdır.Dizi indisleri sıfırdan başlar. Diziler System.Array dediğimiz sınıftan türetilmiştir. Bu sebepten array sınıfının bazı metodlarını dizi işlemlerinde kullanabiliriz.
- Bir dizi oluşturulması iki şekilde yapılabilir, birincisi bildirim ve tanımlamanın aynı satırda yapılması
`int[] dizi=new int[25];`
- İkincisi ise bildirim ve tanımlamanın ayrı satırlarda yapılması.
`int[] dizi;`
`dizi=new int[25];`

Diziler(System.Array)

- new* anahtarı ile tanımlanan nesneler için bir başlangıç değeri atanmaktadır. Referans türleri için NULL, nümerik değerler için 0, bool değerler için false değeridir.

```
int[ ] dizi = { 1, 22, 34 };
```

```
Console.WriteLine(dizi.GetValue(2)); // 34 değerini verir
```

```
Console.WriteLine(dizi[2]); // 34 değerini verir
```

```
string[ ] isimler = { "ali", "ahmet", "selda", "canan", "melike" };
```

```
isimler[0] → "ali"
```

```
dizi[0] → 1
```

Diziler(System.Array)

- *Dizilerin boyutu C ve C++ dillerinde derleme sırasında bilinmek zorundadır. Çünkü dizi için bellekte ne kadarlık bir alan ayrılacağı bilinmelidir. Bu sebepten C ve C++ için dinamik bellek kullanımları mevcuttur.*
- *C# 'ta ise diziler referans tipi olduğu için dizi boyutları çalışma zamanında belirlenebilir.*

```
Console.WriteLine("dizi boyutunu giriniz");  
int boyut=Convert.ToInt32(Console.ReadLine());  
int[] dizi=new int[boyut];
```

Diziler(System.Array)

Diziye rastgele değer ataması;Random sınıfı, System içinde tanımlı.

```
Random rnd = new Random();
int[] dizi = new int[3];
dizi[0] = rnd.Next(2,10); // 2 ile 10 arası rastgele sayı,2 dahil.
dizi[1] = rnd.Next(50); //0 ile 50 arası
dizi[2] = rnd.Next(); //rastgele integer değer
foreach(int i in dizi)           for (k=0;k<3;k++) //0 10 arası değer
{                                { dizi[k] = rnd.Next(10);
    Console.WriteLine(i);        Console.WriteLine(dizi[k]);
}                                }
```

Diziler(Çok boyutlu diziler)

Çok boyutlu dizi tanımı için tanımlama esnasında [] içine verilecek boyut kadar ',' değeri eklenir.

Örnek:

```
int [,] dizi= { {1,3},{3,6},{6,7}};
```

```
int[,] dizi1=new int[2,3];
```

```
dizi1[0,0]=1;
```

```
dizi1[2,1]=7;
```

Diziler(Düzensiz diziler-Jagged)

Öyle bir veri yapısı düşünelim ki , dizi şeklinde olsun,ama bu dizinin içindeki her bir dizinin eleman sayısı farklı olsun. Bu şekildeki dizilere, dizi dizileri denir.

Dizi[0][0]	Dizi[0][1]	Dizi[0][2]	Dizi[0][3]	Dizi[0][4]
Dizi[1][0]	Dizi[1][1]	Dizi[1][2]		
Dizi[2][0]	Dizi[2][1]	Dizi[2][2]	Dizi[2][3]	

```
int [][] dizi= new int[3][];  
dizi[0]=new int[5];  
dizi[1]=new int[3]  
dizi[2]=new int[4]  
dizi.GetLength(0); // satır sayısı  
dizi[2].GetLength(0) // 3. satırdaki sutun sayısı
```



```

int[,] d = {{2,3},{4,5},{6,7}}; // 3x2 lik int dizi.
d[2,1]=33; //doğru
d[2][1]=33; //hata
Console.WriteLine(d.GetValue(2,1)); // 7 değerini yazacak
int[][] dd = new int[2][]; // şekildeki düzensiz dizi oluşturuluyor.ilk önce satırları
dd[0] = new int[2]; //ilk satırın 2 sütunlu olacağı
dd[1] = new int[1]; //ikinci satırın 1 sütunlu olacağı
dd[0][0] = 23;
dd[0][1] = 234;
dd[1][0] = 44;
Random rnd = new Random();
int[] dizi =new int[3];
dizi[0] = rnd.Next(2,10); // 2 ile 10 arası rastgele sayı
dizi[1] = rnd.Next(50); //0 ile 50 arası
dizi[2] = rnd.Next(); //rastgele integer değer
Console.WriteLine(dizi.GetValue(2));//dizi değişkeninin 2. indis elemanını (3.eleman) yazar.
Console.WriteLine(dd[1][0]);
string[] isimler ={ "ali", "ahmet", "selda", "canan", "melike" };
Console.WriteLine("aranan isim=");
string aranan=Console.ReadLine();
aranan.ToLower();
foreach(ss in isimler) {
    if (aranan.Equals(ss)
        Console.WriteLine("aranan isim bulundu...");
        else Console.WriteLine("isim yok");
    }
    Console.ReadKey();

```

23	234
44	

System.Array

Tanımladığımız diziler Array sınıfından türemiş olduklarından dolayı sahip oldukları birtakım metotları ve özellikleri kullanabiliriz.

CreateInstance :

Dizi nesnesi tanımlar.

Array dizi=Array.CreateInstance(typeof(int),3,5); //int tipinde 2 boyutlu

SetValue ve GetValue

Dizi.SetValue(deger,indis) veya Dizi.GetValue(indis) gibi.

System.Array

*Dizileri kopyalamak(**CopyTo** metodu)*

*Bir dizinin tamamını başka bir dizinin istenilen yerine kopyalar.
(kopyalama 0. indexten başlar)*

```
int[] dizi1={1,2,3,4,5,6,7}
```

```
int[] dizi2=new int[8];
```

```
dizi1.CopyTo(dizi2,3); //dizi1 'i dizi2 ye 3. indisten itibaren kopyalar
```

```
Array.Copy(dizi1,2,dizi2,3,1); // dizi1'in 2. indexinden, 1 elemanı,  
dizi2'nin 3. indisinden itibaren kopyalanır.
```

System.Array

*Dizileri sıralama(**Sort** metodu)*

Bir dizinin tamamını sıralamak için kullanılır.

```
Array Dizi=Array.CreateInstance(typeof(string),3);
```

```
Dizi.SetValue("Ali",0);
```

```
Dizi.SetValue("Veli",1);
```

```
Dizi.SetValue("Sami",2);
```

```
Array.Sort(Dizi);
```

System.Array

*Dizileri arama(**BinarySearch** metodu)*

Bir dizide arama yapmak için kullanılır.

```
Array Dizi=Array.CreateInstance(typeof(string),3);  
int indeks;  
Dizi.SetValue("Ali",0);  
Dizi.SetValue("Veli",1);  
Dizi.SetValue("Sami",2);  
string s=Console.ReadLine();  
indeks=Array.BinarySearch(dizi,s); //Array.IndexOf
```

System.Array

Diğer metodlar(Reverse)

Bir diziyi ters çevirir.

```
Array Dizi=Array.CreateInstance(typeof(string),3);  
Dizi.SetValue("Ali",1); //1. indis değerine (2. eleman) Ali atanır  
Dizi.SetValue("Veli",0);  
Dizi.SetValue("Sami",2);  
indeks=Array.Reverse(Dizi);  
indeks=Array.Reverse(Dizi,1,3); //1. elemandan sonraki 3 eleman  
yerlerini ters çevirir.
```

Diğer metodlar(Clear)

Bir dizi elemanını , 0, null ya da false yapar.

```
Array.Clear(Dizi,0,2); // Sıfırıncı elemandan itibaren 2 elemanı temizle
```

System.Collections

ArrayList

Programların çoğunda birden fazla aynı tipte değişkenlere ihtiyaç duyarız. Bu sorunun çözümü olarak birçok dilde kullanılan veri yapıları ,dizilerdir. Bildiğimiz klasik dizilerin programlama tekniklerine getirdikleri kolaylıkların dışında birtakım kısıtlamaları da vardır.

.NET platformunun sınıf kitaplıklarında bulunan ve programcılarının işlerini çok kolaylaştıran ArrayList sınıfı ile klasik dizilerde karşılaştığımız sorunları nasıl çözeceğimizi göreceğiz.

System.Collections

```
using System.Collections; // ArrayList sınıfını kullanmak için
                        // System.Collection isim alanını eklemeliyiz..

class NameSpace
{
    static void Main(string[] args)
    {
        ArrayList aList= new ArrayList();    // aList isimli ArrayList nesnesi oluşturalım.
        // aList nesnemize sırası ile 5, 8, 1 değerlerini        // Add metodu ile ekleyelim.
        aList.Add(5);
        aList.Add(8); aList.Add(1);
        // aList'in elemanlarını ekrana yazdırıyoruz:
        Console.WriteLine("\t aList'in elemanları:");
        foreach(int eleman in aList)
            Console.WriteLine(eleman);
        // aList dizimizden 8 değerini çıkartalım:
        aList.Remove(8);
        // aList dizimize 66 değerini ekleyelim:
        aList.Add(66);
        foreach(int eleman in aList)
            Console.WriteLine(eleman);
        Console.ReadLine();
    }
}
```


System.Collections

Add Bir nesneyi ArrayList'in sonuna ekler.

BinarySearch Sıralanmış bir ArrayList içinde bir nesneyi Binary search algoritması kullanarak arar.

Clear ArrayList'in tüm elemanlarını siler. Sıfırlar.

Contains Herhangi bir nesnenin ArrayList'in elemanı olup olmadığını kontrol eder.

Insert Dizinin sonuna değil de istediğimiz bir yere indeksini belirterek eklememizi sağlar.

Remove Herhangi bir elemanı diziden siler.

Remove At- Belirtilen indisteki elemanı diziden siler.

Reverse Diziyi ters çevirir.

Sort ArrayList'i sıralar.

Count-Eleman sayısı

IndexOf – Kaçınıcı eleman olduğunu bulur...

Capacity kapasiteyi verir yada **set eder**, **TrimToSize()**- Kapasiteyi eleman sayısına azaltır

Çok boyutlu ArrayList

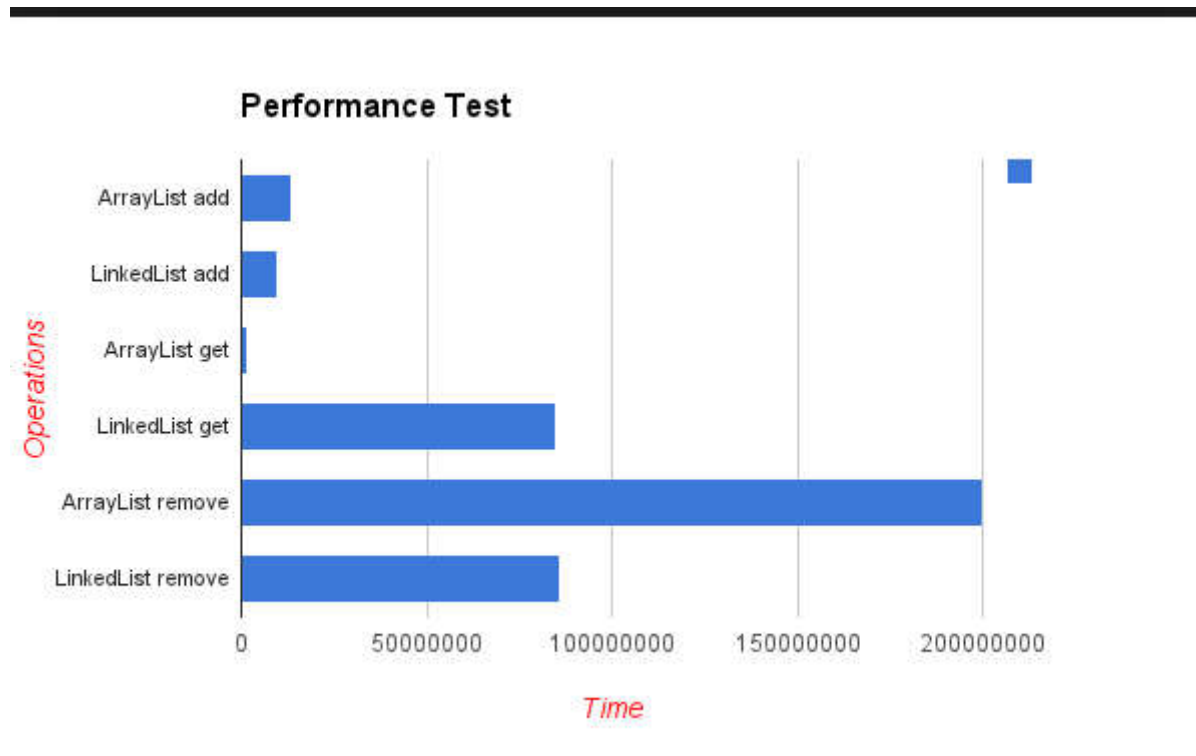
```
liste.Add(new ArrayList());  
    liste.Add(90);  
    (liste[0] as ArrayList).Add(33);
```

Array vs ArrayList

Key Difference Between C# Array vs List

1. Array **stores data of the same sort** whereas ArrayList stores data within the **type of the object which can be of various** sorts.
2. Size of An ArrayList grows **dynamically** whereas Array size remains **static** throughout the program.
3. **Insertion and deletion operation in ArrayList is slower than an Array.**
4. Arrays belong to System. Array namespace whereas ArrayList belongs to System. Collections namespace

Performansları



Dictionary

Dinamik Boyut

Standart diziler sabit boyutludur; programlama aşamasında dizinin boyutu belirtilir ve programın çalışması sırasında değiştirilemez. Dictionary ise değişken boyutludur. Eleman ekleme ve çıkarma durumuna göre boyutu dinamik olarak değişmektedir.

Dictionary Sınıfının Temel Yapısı

Standart dizilere eklenen elemanlar, belleğe sıralı bir şekilde yerleştirilmektedir. Sıfırdan başlanarak her bir elemana birer indeks değeri verilip, elemanlara o indeksler aracılığıyla erişmemiz sağlanmaktaydı. Koleksiyon sınıflarından biri olan **ArrayList** içinde aynı durum söz konusu. ArrayList'e eklenen her bir elemana indeks numarasıyla erişebilmekteyiz.

Dictionary

Dictionary koleksiyonunda ise Anahtar(Key) ve Değer(Value) olmak üzere iki kavram karşımıza çıkmakta. Konuyu daha anlaşılır kılmak açısından; standart dizilere eklediğimiz elemanları Değer, o elemanlara erişmek için kullandığımız indeksleri de Anahtar olarak düşünebilirsiniz.

Her bir Değerin farklı bir Anahtarı olmalıdır yani koleksiyon içerisinde yer alan Anahtarlar birbirinden farklı olmalıdır.

Dictionary sınıfından bir nesne oluştururken, anahtar ve değer veri tiplerini belirtmemiz gerekmekte. Aşağıdaki kod satırında, Key_Tipi yerine Anahtarın veri tipini, Value_Tipi yerine de Değerin verini tipini belirtmeliyiz.

```
Dictionary<Key_Tipi, Value_Tipi> Referans_Adi = new Dictionary<Key_Tipi, Value_Tipi>();
```

Dictionary

```
C:\WINDOWS\system32\cmd.exe
0-->kocaeli
1-->True
2-->uni
3-->77
[0, kocaeli]
[1, True]
[2, uni]
[3, 77]
Press any key to continue . . .
```

```
static void Main(string[] args)
{
    Dictionary<int, object> kullanici = new Dictionary<int, object>();
    kullanici.Add(0, "kocaeli");
    kullanici.Add(1, true);
    kullanici.Add(2, "uni");
    kullanici.Add(3, 77);

    foreach (KeyValuePair<int, object> item in kullanici)
    {
        Console.WriteLine(item.Key + "-->" + item.Value);
        //MessageBox.Show(item.Key.ToString()+"--"+item.Value.ToString());
    }
    foreach (var t in kullanici)
    {
        Console.WriteLine(t);
    }
}
```

Dictionary

Metotlar ve Özellikler

- **ContainsKey(Aranan_Key) Metodu**

Koleksiyon içerisinde, parametre olarak girilen değerde bir Anahtar (Key) mevcutsa TRUE değilse FALSE döndürecektir.

```
bool varmi = Ogranci.ContainsKey(158);
```

- **ContainsValue(Aranan_Value) Metodu**

Koleksiyon içerisinde, parametre olarak girilen değerde bir Değer (Value) mevcutsa TRUE değilse FALSE döndürecektir.

```
bool varmi = Ogranci.ContainsValue("Kadir Aydemir");
```

- **Clear() Metodu**

Koleksiyon içerisinde yer alan tüm Anahtar-Değer çiftlerini silmektedir.

```
Ogranci.Clear();
```


Dictionary

Metotlar ve Özellikler

- **Count Özelliği**

Koleksiyon içinde bulunan anahtar/değer çiftlerinin sayısını döndürmektedir.

```
int ElemanSayisi = Ogrenci.Count;
```

- **Remove(Silinecek_Key) Metodu**

Koleksiyon içerisinde, parametre olarak girilen değerde bir Anahtar (Key) mevcutsa; Anahtarı ve anahtarla ilişkili Değeri silip TRUE döndürecektir. Anahtar mevcut değilse FALSE döndürecektir.

```
bool silindiMi=kullanici.Remove(43);  
    if(!silindiMi)  
    {  
        Console.WriteLine("silme olmadı");  
    }
```

Dictionary

Metotlar ve Özellikler

```
0  
1  
2  
3  
kocaeli  
True  
uni  
77  
Press any key to continue . . .
```

- **Keys Özelliği**
Anahtarları (Keys) içeren bir koleksiyon döndürmektedir.
- **Values Özelliği**
Değerleri (Values) içeren bir koleksiyon döndürmektedir.

```
//anahtarları almak  
Dictionary<int, object>.KeyCollection anahtarlar = kullanici.Keys;  
foreach (var item in anahtarlar)  
{  
    Console.WriteLine(item);  
}  
  
//değerleri almak  
Dictionary<int, object>.ValueCollection degerler = kullanici.Values;  
foreach (var item in degerler)  
{  
    Console.WriteLine(item);  
}
```

Dictionary

Metotlar ve Özellikler

```
0  
1  
2  
3  
kocaeli  
True  
uni  
77  
Press any key to continue . . .
```

For ile ulaşmak (ElementAt kullanımı)

```
for (int i = 0; i < kullanici.Count-1; i++)  
{  
    Console.WriteLine(kullanici.ElementAt(i).Value);  
}
```

List<T>

List < T > Class Nedir?

Koleksiyon sınıfları özel tasarlanmış nesneleri ve onlara ait olan görevleri yerine getirmek için oluşturulmuş olan nesnelerdir. List Class System.Collections.Generic isim uzayı içinde tanımlanmış metodlar, özellikler ve diğer sınıflarda olduğu gibi insert, remove, search vb. nesneleri barındırmaktadır. List class diziler(array) ve veri yapıları (data structure) nesneleri yerine kullanılır. List sınıfları kullanıldığında dizi üzerindeki boyutundaki esneklik yanı sıra ek özellikleri de kolaylık sağlamaktadır.

C# List < T > sınıfı nesnelerin türünü oluşturulduğunda belirtme zorunluluğu göstermektedir.

List < T > Kullanımı; T parametresi listedeki nesnelerin türünü ifade etmektedir.

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

Performans

Collection	Ordering	Contiguous Storage?	Direct Access?	Lookup Efficiency	Manipulate Efficiency	Notes
Dictionary	Unordered	Yes	Via Key	Key: $O(1)$	$O(1)$	Best for high performance lookups.
SortedDictionary	Sorted	No	Via Key	Key: $O(\log n)$	$O(\log n)$	Compromise of Dictionary speed and ordering, uses binary search tree.
SortedList	Sorted	Yes	Via Key	Key: $O(\log n)$	$O(n)$	Very similar to SortedDictionary, except tree is implemented in an array, so has faster lookup on preloaded data, but slower loads.
List	User has precise control over element ordering	Yes	Via Index	Index: $O(1)$ Value: $O(n)$	$O(n)$	Best for smaller lists where direct access required and no sorting.
LinkedList	User has precise control over element ordering	No	No	Value: $O(n)$	$O(1)$	Best for lists where inserting/deleting in middle is common and no direct access required.
HashSet	Unordered	Yes	Via Key	Key: $O(1)$	$O(1)$	Unique unordered collection, like a Dictionary except key and value are same object.
SortedSet	Sorted	No	Via Key	Key: $O(\log n)$	$O(\log n)$	Unique sorted collection, like SortedDictionary except key and value are same object.
Stack	LIFO	Yes	Only Top	Top: $O(1)$	$O(1)^*$	Essentially same as List<T> except only process as LIFO
Queue	FIFO	Yes	Only Front	Front: $O(1)$	$O(1)$	Essentially same as List<T> except only process as FIFO