

DATA STRUCTURES

Part – IV

IEEE 754 FP Standardı, Dinamik Dizi,
Koleksiyonlar, Diğer Önemli Konular ve İlgili
Programlama Örnekleri

IEEE 754 Standard for Floating-Point Arithmetic

İkiye Tümleyen (Two's Complement)

İkiye tümleyen, negatif sayıları ikili sayılar içerisinde göstermek için kullanılan bir yöntemdir. İlave bir $+$ ve $-$ sembolüne ihtiyaç duyulmaz. İkili (binary) bir sayının ikiye tümleyeni (two's complement), sayının ikinin üssünden çıkartılması ile elde edilir (n bitlik ikinin tümleyeni için, 2^N 'den çıkarılır).

Negatif sayıların ikili kodlanması esnasında ikiye tümleme metodu kullanılmadığı takdirde, sayının işareti için ayrı bir bit harcanması gerekir. Birbirinden farklı işarete sahip birden fazla 0 sayısının oluşmasına sebep olur. Bitlerin değerlerini ve anlamlarını değerlendirecek bir yönetim-denetim mekanizması kullanılması gerekir.

8 Bitlik İkinci Tümüleyeni Tamsayılar

A **two's-complement system** or **two's-complement arithmetic** is a system in which negative numbers are represented by the two's complement of the absolute value.

this system is the most common method of representing signed integers on computers. In such a system, a number is negated (converted from positive to negative or vice versa) by computing its two's complement.

Pozitif Sayılar doğal şekildedir:

most-significant bit								
0	1	1	1	1	1	1	1	= 127
0	1	1	1	1	1	1	0	= 126
0	0	0	0	0	0	1	0	= 2
0	0	0	0	0	0	0	1	= 1
0	0	0	0	0	0	0	0	= 0
1	1	1	1	1	1	1	1	= -1
1	1	1	1	1	1	1	0	= -2
1	0	0	0	0	0	0	1	= -127
1	0	0	0	0	0	0	0	= -128

8-bit two's-complement integers

Sınırları : $[-2^{N-1} \dots 2^{N-1}-1]$

Java *Byte* Data Type (=C# *sbyte* Data Type)

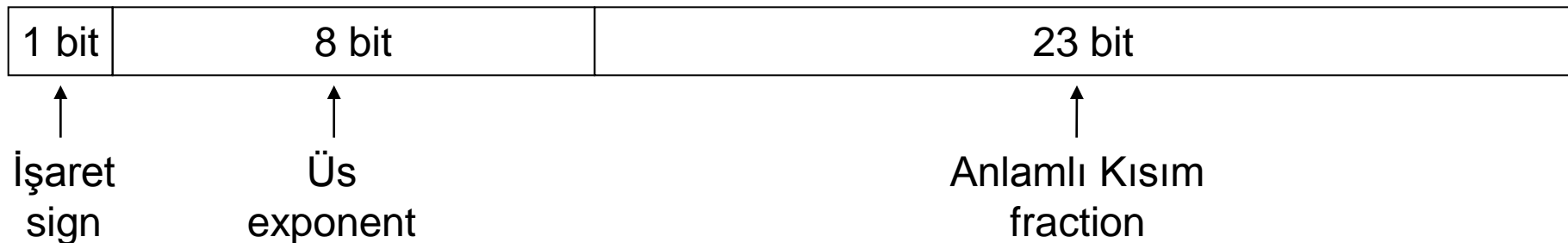
Decimal	8 bit Binary	Invert	add 1 for Twos Complement	Decimal
0	0000 0000	-	-	-
+1	0000 0001	1111 1110	1111 1111	-1
+2	0000 0010	1111 1101	1111 1110	-2
+3	0000 0011	1111 1100	1111 1101	-3
+4	0000 0100	1111 1011	1111 1100	-4
+5	0000 0101	1111 1010	1111 1011	-5
—				—
124	0111 1100	1000 0011	1000 0100	-124
125	0111 1101	1000 0010	1000 0011	-125
126	0111 1110	1000 0001	1000 0010	-126
127	0111 1111	1000 0000	1000 0001	-127
-	-	-	1000 0000	-128

IEEE 754 Standard for Floating-Point Arithmetic

- http://tr.wikipedia.org/wiki/IEEE_754
- http://en.wikipedia.org/wiki/IEEE_754-1985
- **IEEE Kayan Nokta Aritmetiği Standardı**, kayan noktalı sayıların gösteriminde en çok kullanılan standarttır.
- **IEEE 754** standardına göre sayılar tek duyarlı (32 bit) ve çift duyarlı(64 bit) şekilde gösterilebilirler.

IEEE 754 : Tek Duyarlı Gösterim

Tek duyarlı gösterimde sayı 32 bitle ifade edilir. Bu bitlerden biri işaret, 8'i üs 23 tanesi ise anlamlı kısmın gösterimi için kullanılır. Tek duyarlı gösterimde üs için kaydırma değeri $2^{8-1}-1 = 127$ olarak hesaplanır.



IEEE 754 : Tek Duyarlı Gösterim

Tek duyarlı gösterimde 6,375 sayısını temsil etmek istersek :

$$6 \rightarrow (110)_2$$

$$0,375 \times 2 = 0,75$$

$$0,75 \times 2 = 1,5$$

$$0,5 \times 2 = 1,0$$

$$0,375 = (0,011)_2 \rightarrow 6,375 = (110,011)_2$$

Sayıyı olağan duruma getirirsek

$$: 110,011 = 1,10011 \times 2^2$$

Sayı > 0 olduğundan işaret biti

$$: 0$$

Sayının üst değerinin saptırılmış hali

$$: 2+127 = 129 \rightarrow 129_{10} = 10000001_2$$

Anlamlı kısım

$$: 100110000000000000000000$$

Sayı son olarak

$$: 0 \ 10000001 \ 100110000000000000000000$$

şeklinde ifade edilir.

IEEE Tek Duyarlı Gösterimdeki Bir Sayının Onluk Düzendeki Değerinin Bulunması

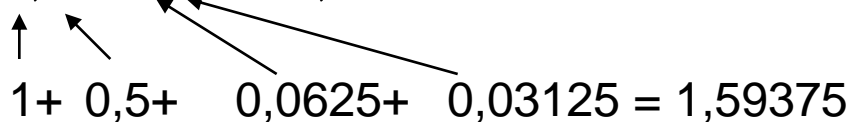
$$v = s \times 2^e \times m$$

where

- $s = +1$ (positive numbers and +0) when the sign bit is 0
- $s = -1$ (negative numbers and -0) when the sign bit is 1
- $e = \text{exponent} - 127$ (in other words the exponent is stored with 127 added to it, also called "biased with 127")
- $m = 1.\text{fraction}$ in binary (that is, the significand is the binary number 1 followed by the radix point followed by the binary bits of the *fraction*). Therefore, $1 \leq m < 2$. (1+fraction)

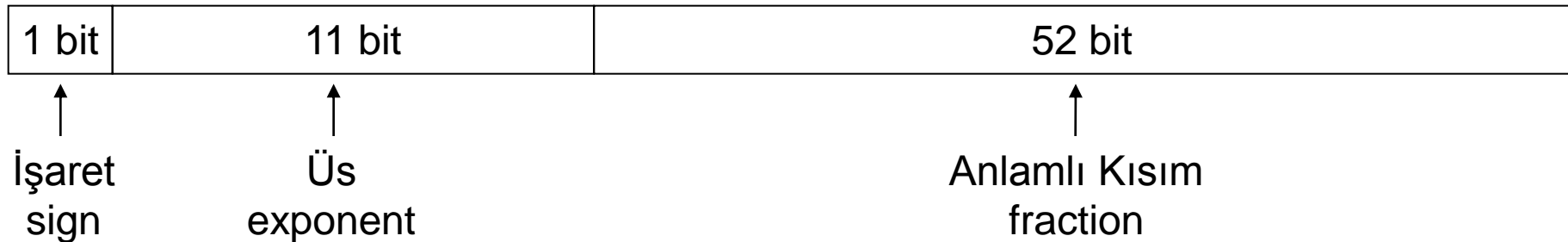
0 10000001 100110000000000000000000

$$v = +1 * 2^2 * (1,10011)_2 = 6,375$$


$$1+ 0,5+ 0,0625+ 0,03125 = 1,59375$$

IEEE 754 : Çift Duyarlı Gösterim

Çift duyarlı gösterimde sayı 64 bitle ifade edilir. Bu bitlerden biri işaret, 11'i üs ve 52 tanesi de anlamlı kısmı ifade etmek için kullanılır. Bu gösterimde üs için sapma değeri $2^{11-1}-1 = 1023$ olarak hesaplanır.



Object Sınıfı: Tüm sınıflar ve tipler object sınıfından türetilmiştir.

Kutulama (Boxing) : Değer tipindeki verinin referansa çevrilmesi

Kutudan Çıkarma (Unboxing) : Değerin kutu dışına kopyalanması

Types Conversions

- ◆ Implicit conversions
 - Occur automatically
 - Guaranteed to succeed
 - No information (precision) loss
- ◆ Explicit conversions
 - Require a cast
 - May not succeed
 - Information (precision) might be lost
- ◆ Both implicit and explicit conversions can be user-defined

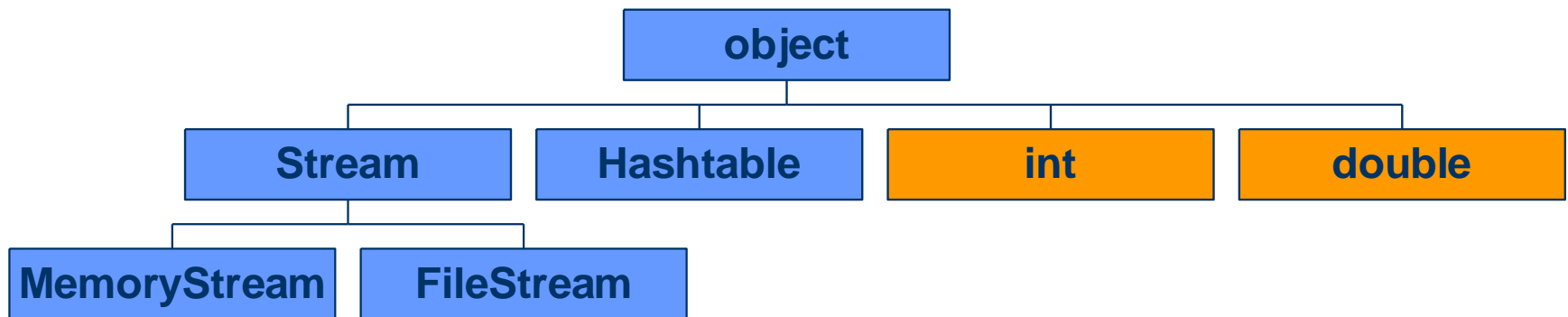
Types Conversions

```
int x = 123456;  
long y = x;           // implicit  
short z = (short)x;   // explicit  
  
double d = 1.2345678901234;  
float f = (float)d;    // explicit  
long l = (long)d;      // explicit
```

Types

Unified Type System

- ◆ Everything is an object
 - All types ultimately inherit from object
 - Any piece of data can be stored, transported, and manipulated with no extra work



Types

Unified Type System

◆ Polymorphism

- The ability to perform an operation on an object without knowing the precise type of the object

```
void Poly(object o) {  
    Console.WriteLine(o.ToString());  
}
```

```
Poly(42);  
Poly("abcd");  
Poly(12.345678901234m);  
Poly(new Point(23,45));
```

Types

Unified Type System

- ◆ Question: How can we treat value and reference types polymorphically?
 - How does an int (value type) get converted into an object (reference type)?
- ◆ Answer: Boxing!
 - Only value types get boxed
 - Reference types do not get boxed

Types

Unified Type System

◆ Boxing

- Copies a value type into a reference type (object)
- Each value type has corresponding “hidden” reference type
- Note that a reference-type copy is made of the value type
- Value type is converted implicitly to object, a reference type
 - Essentially an “up cast”

Types

Unified Type System

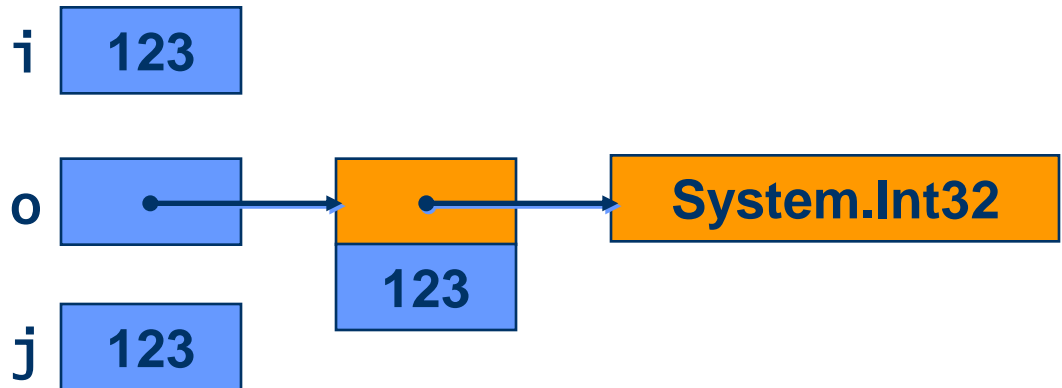
- ◆ Unboxing
 - Inverse operation of boxing
 - Copies the value out of the box
 - Copies from reference type to value type
 - Requires an explicit conversion
 - May not succeed (like all explicit conversions)
 - Essentially a “down cast”

Types

Unified Type System

- ◆ Boxing and unboxing

```
int i = 123;  
object o = i;  
int j = (int)o;
```



Types

Unified Type System

- ◆ Benefits of boxing
 - Enables polymorphism across all types
 - Collection classes work with all types
 - Eliminates need for wrapper classes
- ◆ Lots of examples in .NET Framework

```
Hashtable t = new Hashtable();  
t.Add(0, "zero");  
t.Add(1, "one");  
t.Add(2, "two");
```

```
string s = string.Format(  
    "Your total was {0} on {1}",  
    total, date);
```

Dinamik Dizi (ArrayList)

Dinamik Dizi

- Dinamik dizinin boyutu değiştirilebilir. Genişletilip, daraltılabilir yani eleman sayısı artırılıp azaltılabilir.
- Araya eleman eklemek, aradan eleman çıkarmak kolay ve etkindir. Elemanları kaydırmak için kod yazmaya gerek kalmaz.
- Her veri tipinde nesneyi (elemanı) tutabilir.

Bazı Dinamik Dizi Metotları

Public Properties [Capacity](#)

Gets or sets the number of elements that the **ArrayList** can contain.

[Count](#)

Gets the number of elements actually contained in the **ArrayList**.

Public Methods

[Add](#).

Adds an object to the end of the **ArrayList**.

[BinarySearch](#)

Overloaded. Uses a binary search algorithm to locate a specific element in the sorted **ArrayList** or a portion of it.

[Clear](#)

Removes all elements from the **ArrayList**.

[Clone](#)

Creates a shallow copy of the **ArrayList**.

[Contains](#)

Determines whether an element is in the **ArrayList**.

[CopyTo](#)

Overloaded. Copies the **ArrayList** or a portion of it to a one-dimensional array.

[Equals](#)

Overloaded. Determines whether two [Object](#) instances are equal.

[GetRange](#)

Returns an **ArrayList** which represents a subset of the elements in the source **ArrayList**.

[GetType](#)

Gets the [Type](#) of the current instance.

[IndexOf](#)

Overloaded. Returns the zero-based index of the first occurrence of a value in the **ArrayList** or in a portion of it.

[Insert](#)

Inserts an element into the **ArrayList** at the specified index.

[InsertRange](#)

Inserts the elements of a collection into the **ArrayList** at the specified index.

[LastIndexOf](#)

Overloaded. Returns the zero-based index of the last occurrence of a value in the **ArrayList** or in a portion of it.

[Remove](#)

Removes the first occurrence of a specific object from the **ArrayList**.

[RemoveAt](#)

Removes the element at the specified index of the **ArrayList**.

[RemoveRange](#)

Removes a range of elements from the **ArrayList**.

[Reverse](#)

Overloaded. Reverses the order of the elements in the **ArrayList** or a portion of it.

[Sort](#)

Overloaded. Sorts the elements in the **ArrayList** or a portion of it.

[ToArray](#)

Overloaded. Copies the elements of the **ArrayList** to a new array.

[ToString](#)

Returns a [String](#) that represents the current [Object](#).

[TrimToSize](#)

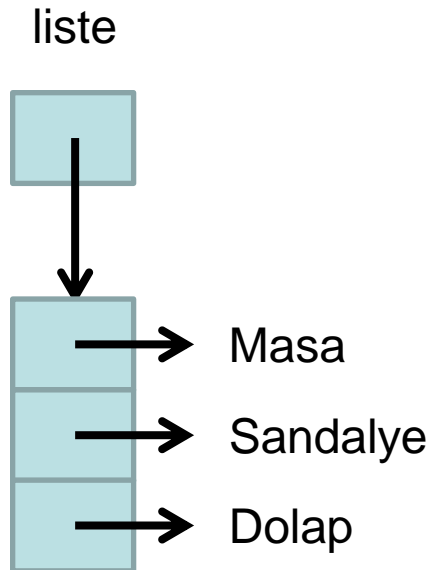
Sets the capacity to the actual number of elements in the **ArrayList**.

Örnek 1

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

Ekran Çıktısı :
Masa
Sandalye
Dolap

```
namespace ConsoleApplication18  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            ArrayList liste = new ArrayList();  
            liste.Add("Masa");  
            liste.Add("Sandalye");  
            liste.Add("Dolap");  
            foreach (Object o in liste)  
                Console.WriteLine(o);  
        }  
    }  
}
```



Örnek 2

Değişik tiplerde veri eklenmesi.

```
static void Main(string[] args)
{
    ArrayList liste = new ArrayList();
    liste.Add("Masa");
    liste.Add('c');
    liste.Add(5.5);
    foreach (Object o in liste)
        Console.WriteLine(o);
}
```

Ekran Çıktısı :

Masa

c

5,5

```
foreach (string s in liste)
    Console.WriteLine(s);
```

yazılsa burada hata verir. 2 elemanı
string değil.

ArrayList Metotlarının Kullanımları - I

```
ArrayList liste = new ArrayList();  
liste.Add("Masa");  
liste.Add("Sandalye");  
liste.Add("Dolap");  
liste.Insert(1, "Sehpa");
```

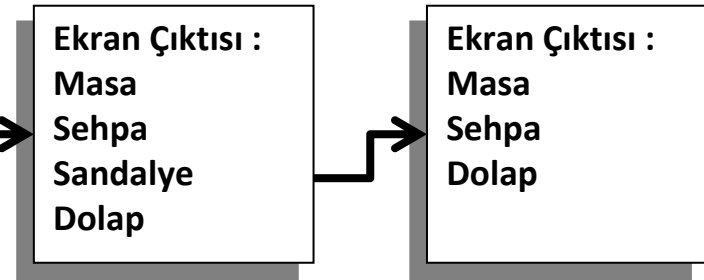
Ekran Çıktısı :

Masa
Sehpa
Sandalye
Dolap

```
ArrayList liste2 = new ArrayList(10);  
liste2.AddRange(liste);  
foreach (Object o in liste2)  
    Console.WriteLine(o);  
Console.WriteLine(liste2.Count+" "+liste2.Capacity); // 4 ve 10
```

ArrayList Metotlarının Kullanımları - II

```
liste.RemoveAt(2);  
foreach (Object o in liste)  
    Console.WriteLine(o);
```



`Console.WriteLine(liste[2]);` -> Dolap

Elemanları diziye aktarma, verilen bir elemanın indisini bulma, tümünü silme ve sıralama metotlarını deneyiniz.

ArrayList Metotlarının Kullanımları - III

```
System.Collections.ArrayList liste = new  
    System.Collections.ArrayList();
```

```
liste.Add("Masa");  
liste.Add("Sandalye");  
liste.Add("Dolap");  
liste.Insert(1, "Sehpa");
```

using Collections; demezseniz
alternatif kullanım.

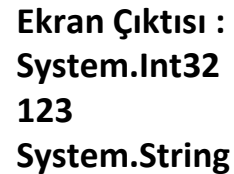
Elemanlardan birisi bile string değilse
hata verir! liste.Insert(1, "Sehpa"); yerine
liste.Insert(0, 123); deseydik
InvalidCastException
Unable to cast object of type
'System.Int32' to type 'System.String'.
mesajı gelirdi.

```
liste.RemoveAt(1);  
liste.Remove("Dolap");  
foreach(string str in liste)  
    Console.WriteLine(str);
```

Ekran Çıktısı :
Masa
Sandalye

Veri Tipi Belirleme

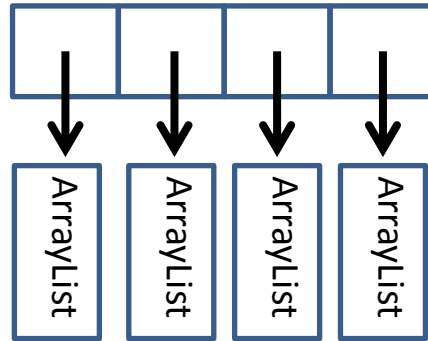
```
liste.Add("Masa");  
liste.Add("Sandalye");  
liste.Add("Dolap");  
liste.Insert(0, 123);  
liste.RemoveAt(1);  
liste.Remove("Dolap");
```



Ekran Çıktısı :
System.Int32
123
System.String

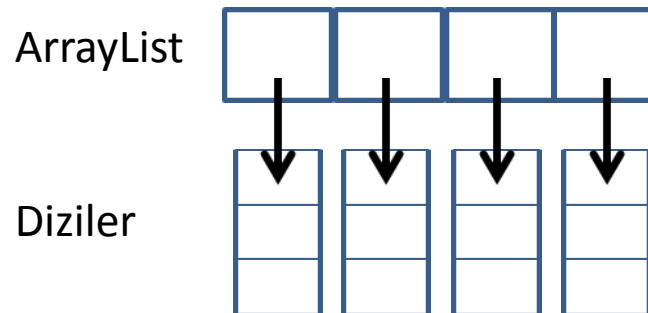
```
foreach (Object obj in liste)  
{  
    Console.WriteLine(obj.GetType());  
    if (obj.GetType().ToString() == "System.Int32")  
        Console.WriteLine(obj);  
}
```

ArrayList Dizisi oluşturunuz!



```
ArrayList[] a1 = new ArrayList[4];  
a1[0] = new ArrayList();  
a1[0].Add("Merhaba");  
.....
```

Dizilerden Oluşan Bir ArrayList tanımlayınız.



```
ArrayList al = new ArrayList();  
int[] dizi = new int[3];  
dizi[0] = 5; dizi[1] = 5;  
al.Add(dizi);
```

Düşününüz

- Dizilerden oluşan bir dizi.
- ArrayList'lerden oluşan ArrayList.

Hash Tablosu

- Anahtar karma kodu göre düzenlenen anahtar/değer çiftleri topluluğu temsil eder.
- Anahtarları değerlere eşleyen hash fonksiyonu kullanan veri yapısıdır. Hızlı erişim sağlar.
- <http://msdn.microsoft.com/tr-tr/library/system.collections.hashtable.aspx>

Örnek 3

```
static void Main(string[] args)
{
    Hashtable maaş = new Hashtable();
    maaş[3] = 1000;
    maaş[5] = 1500;
    maaş[15] = 900;
```

5, 1500
3, 1000
15, 900
1000

maaş[key] = value;
Key ve value,
object olabilir.

```
foreach (DictionaryEntry entry in maaş)
    Console.WriteLine("{0}, {1}", entry.Key, entry.Value);
```

```
Console.WriteLine(maaş[3]);
```

```
}
```

maaş.Remove(5); ?

Elemanları tek tek dolaşma: 1500, 1000, 900

```
IDictionaryEnumerator enumerator = maaş.GetEnumerator();
while (enumerator.MoveNext())
    Console.WriteLine(enumerator.Value);
```

static anahtar kelimesi ve sınıf üyeleri

Bu static nedir? (Deitel'den)

- Each object of a class has its own copy of all the instance variables of the class.
- However, in certain cases, all class objects should share only one copy of a particular variable.
- Such variables are called ***static variables***.
- A program contains only one copy of each of a class's static variables in memory, no matter how many objects of the class have been instantiated.

Örnek 4

```
class Öğrenci
{
    public Öğrenci() { sayı ++; }
    public static int sayı;
    string ad;
}
```

Static sınıf üyeleri, sınıfın bir örneği (nesnesi) oluşturulmadan, veri ve fonksiyon oluşturmak için kullanılırlar.

Nesneden bağımsız veri ve metot oluşturulur.

```
class Program
{
    static void Main(string[] args)
    {
        Öğrenci ogr1 = new Öğrenci();
        Öğrenci ogr2 = new Öğrenci();
        Console.WriteLine(Öğrenci.sayı); // 2
    }
}
```

Öğrenci sınıfı için sadece 1 adet sayı değişkeni tutulacağından, nesnelerin sayısı burada hesaplanabilir. Her bir nesne için ayrıca oluşturulmaz.

ad static tanımlanmadığından, her bir nesne için birer ad sahası oluşturulur.

Main neden static'tir

- Main, programın kapısıdır.
- Nesne oluşturmada çağrılması gerekir, derleyiciden.
- Diğer static metotlar da, new ile sınıfın bir örneği yani nesnesi oluşturulmadan çağrılabilirler.

Koleksiyonlar (veya Topluluklar)

Collections

System.Collections isim uzayı, çok sayıda sınıf ve arayüz içerir.

Hazır Veri Yapıları olarak düşünülebilirler.

Programlama sırasında, gerçekleştirimlerinin nasıl yapıldığının bilinmesi gerekmeden kullanılabilirler.

Stack and Queue

Do you need a sequential list where the element is typically discarded after its value is retrieved?

- If yes, consider using the [Queue](#) class or the [Queue](#) generic class if you need first-in-first-out (FIFO) behavior. Consider using the [Stack](#) class or the [Stack](#) generic class if you need last-in-first-out (LIFO) behavior.
- If not, consider using the other collections.

Access Order

Do you need to access the elements in a certain order, such as FIFO, LIFO, or random?

- The **Queue** class and the **Queue** generic class offer FIFO access.
- The **Stack** class and the **Stack** generic class offer LIFO access.
- The [LinkedList](#) generic class allows sequential access either from the head to the tail or from the tail to the head.
- The rest of the collections offer random access.

Collections offer random access

Do you need to access each element by index?

- The [ArrayList](#) and [StringCollection](#) classes and the [List](#) generic class offer access to their elements by the zero-based index of the element.
- The [Hashtable](#), [SortedList](#), [ListDictionary](#), and [StringDictionary](#) classes, and the [Dictionary](#) and [SortedDictionary](#) generic classes offer access to their elements by the key of the element.
- The [NameObjectCollectionBase](#) and [NameValueCollection](#) classes, and the [KeyedCollection](#) and [SortedList](#) generic classes offer access to their elements by either the zero-based index or the key of the element.

Arayüzler (Interfaces)

- Koleksiyon sınıfları için ortak olan işlevsellikler, arayüzler tarafından belirlenir.
- `ICollection` arayüzünde, tüm koleksiyonların sahip olması gereken metot ve özellikler tanımlanmıştır.
- Örnek olarak `int Count`, koleksiyondaki eleman sayısını tutar.
- `void copyTo (Array hedef, int strtIndis)` metodu, elemanları hedef diziye kopyalar.

IEnumerable, IList ve IDictionary

- IEnumerable, kalıtım yolu ile ICollection'a aktarılmaktadır. Enumeration -> Birer birer saymak için kullanılır.
- IList, kalıtımla ICollection'dan türetilir. Elemanlarına bir indeks aracılığı ile erişilmesine izin verir. add, insert, remove gibi metotları vardır.
- IDictionary, benzersiz anahtarlı yapılarda elemanlara erişilmesine izin verir. A dictionary is an associative array.

Generics

- C# generics are like C++ *templates*.
- We create a specialized version for a specific type by supplying the name of the type when the class is used.
- All of the containers so far have stored data as objects. This means
 - Containers hold any type
 - Operations must test the type before operating on the objects in the container
 - When objects are removed from the container they must be cast to their true type
 - Introduces new potential sources of error
 - Forces expensive run-time type checking

Genel Tipler (Generics)

- Sınıfı object için değil, istediğimiz belli bir veri tipi için oluştururuz.
- Kutulama ve kutudan çıkarmaya gerek kalmaz.

Örnek 5

List <T> , generic List sınıfıdır.

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

```
liste.Add(1);
```

```
liste.Add(2);
```

```
liste.Add(3);
```

```
foreach (int i in liste)
```

```
{
```

```
    Console.WriteLine(i);
```

```
}
```

Örnek 6

```
ArrayList liste = new ArrayList();
```

```
liste.Add("aaa");
```

```
liste.Add(2);
```

```
liste.Add(3);
```

```
foreach (int i in liste)
```

```
{
```

```
    Console.WriteLine(i);
```

```
}
```

Generic yapmazsak, her veri tipini alır ancak burada **Casting** hatası verir.

Data Structures dersi

kapsamında anlatılanlar

- Veri Tipleri ve Değişkenler
- Değer Tipleri ve Referans Tipleri
- Veri Tipi Dönüşümleri
- Diziler ve Çok Boyutlu Diziler
- Düzensiz Diziler
- Sınıflar ve Nesneler (Yeni Veri Tipi Oluşturma) ve Veri Yapısı Oluşturmanın Temeli
- Veriler Bilgisayarda Nasıl Temsil Ediliyor? IEEE 754
- ArrayList, Bileşik Yapılar ve Koleksiyonlar

Geçici Bilgi Tutmak

```
Object o;
```

```
for (int i = 0; i < 10; ++i)  
    o = Console.ReadLine();
```

Bilgiyi saklamak gerekmiyorsa, tek referans çok sayıda bilgi okumak için kullanılabilir.

Metotlar

Slides and Reading bölümündeki sunumu
inceleyiniz.

Solution Explorer

- Projelerin, dosyaların ve kullanıma hazır ilgili komutların düzenlenmiş bir görünümünü sunar.
- .cs uzantılı dosyaya çift tıklandığında sadece ilgili kaynak kod gelir.
- Projenizi açmak için .sln uzantılı Solution dosyasına çift tıklamak yeterlidir.

İççe Aduzayları (Nested Namespaces) ve İççe Sınıflar (Nested Classes)

Solution Explorer'da ilgili Uygulama üzerinde sağ tuşa basılıp
Add -> New Item -> Class denilerek projeye yeni sınıflar eklenebilir.

```
namespace ConsoleApplication24
{
    class Program
    {
        static void Main(string[] args)
        {
            A.B.Class1 sınıf = new A.B.Class1();
        }
    }
}
```

```
namespace A
{
    namespace B
    {
        class Class1
        {
        }
    }
}
```


Program Structure

Namespaces

- ◆ The using statement lets you use types without typing the fully qualified name
- ◆ Can always use a fully qualified name

```
using N1;

C1 a;           // The N1. is implicit
N1.C1 b;        // Fully qualified name

C2 c;           // Error! C2 is undefined
N1.N2.C2 d;     // One of the C2 classes
C1.C2 e;        // The other one
```

Program Structure

Namespaces

- ◆ The `using` statement also lets you create aliases

```
using C1 = N1.N2.C1;  
using N2 = N1.N2;
```

```
C1 a;           // Refers to N1.N2.C1  
N2.C1 b;        // Refers to N1.N2.C1
```


Kaynaklar

- ◆ Data Structures Lecture Notes, Dr. Aybars UĞUR
- ◆ Introduction to C# Lecture Notes, Mark Sapossnek
- C# 4.0 Herkes İçin, Herbert Schildt, Alfa Yayınları, 2011.